NONLINEAR OPTIMIZATION

Qingsha Cheng 程庆沙

Surrogate-Based Modeling and Optimization I:

Surrogate-based optimization: motivation and concept

Surrogate modeling flowchart

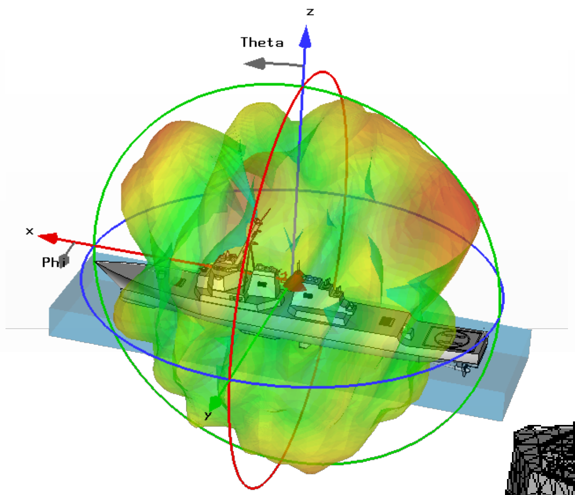
Design of experiments

Surrogate-Based Optimization: Motivation and Concept

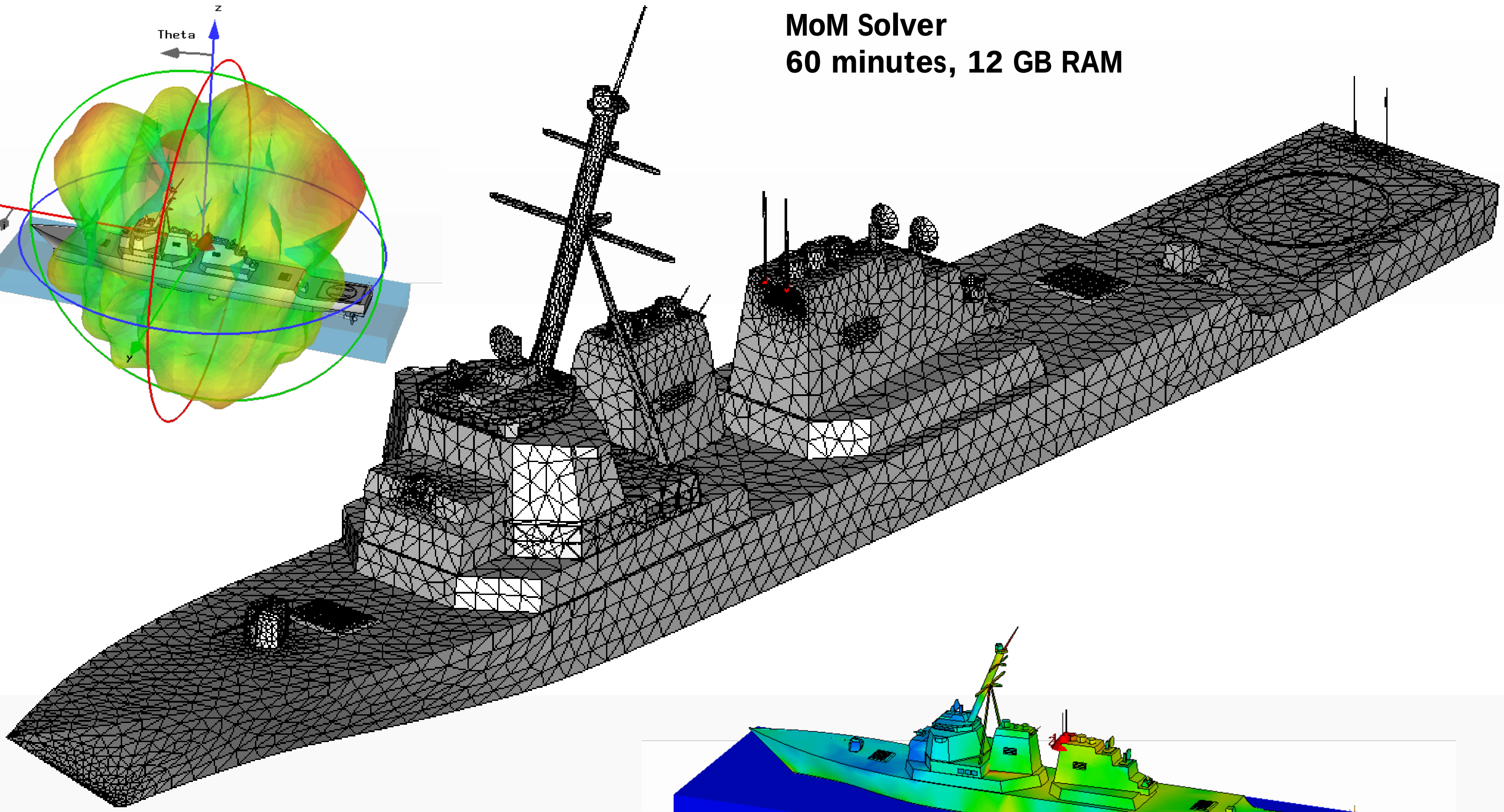
1. Objective function is very expensive
2. Objective function is non-differentiable or even discontinuous
3. Objective function is noisy
4. Derivative information is not available
5. Objective function can only be evaluated at a (sometimes even finite) subset of the domain

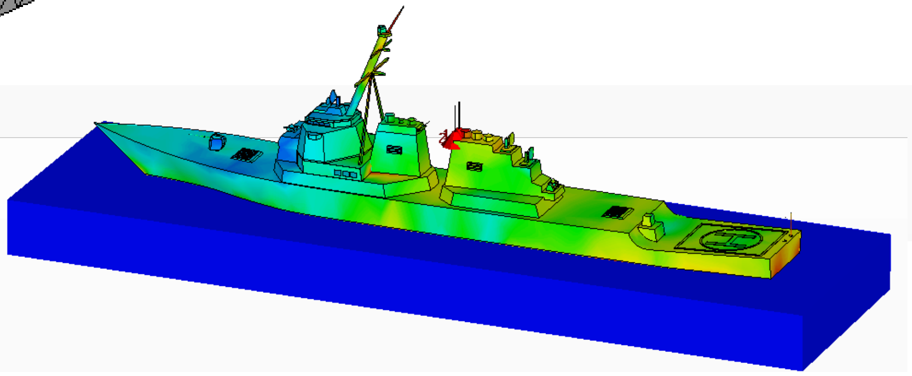
=> Using direct (especially gradient-based) optimization methods is prohibitive

HF ANTENNAS (10 MHZ)

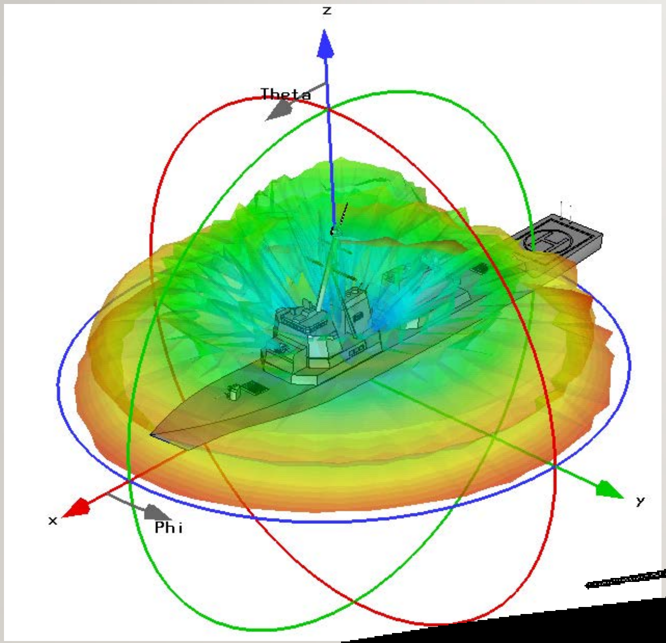


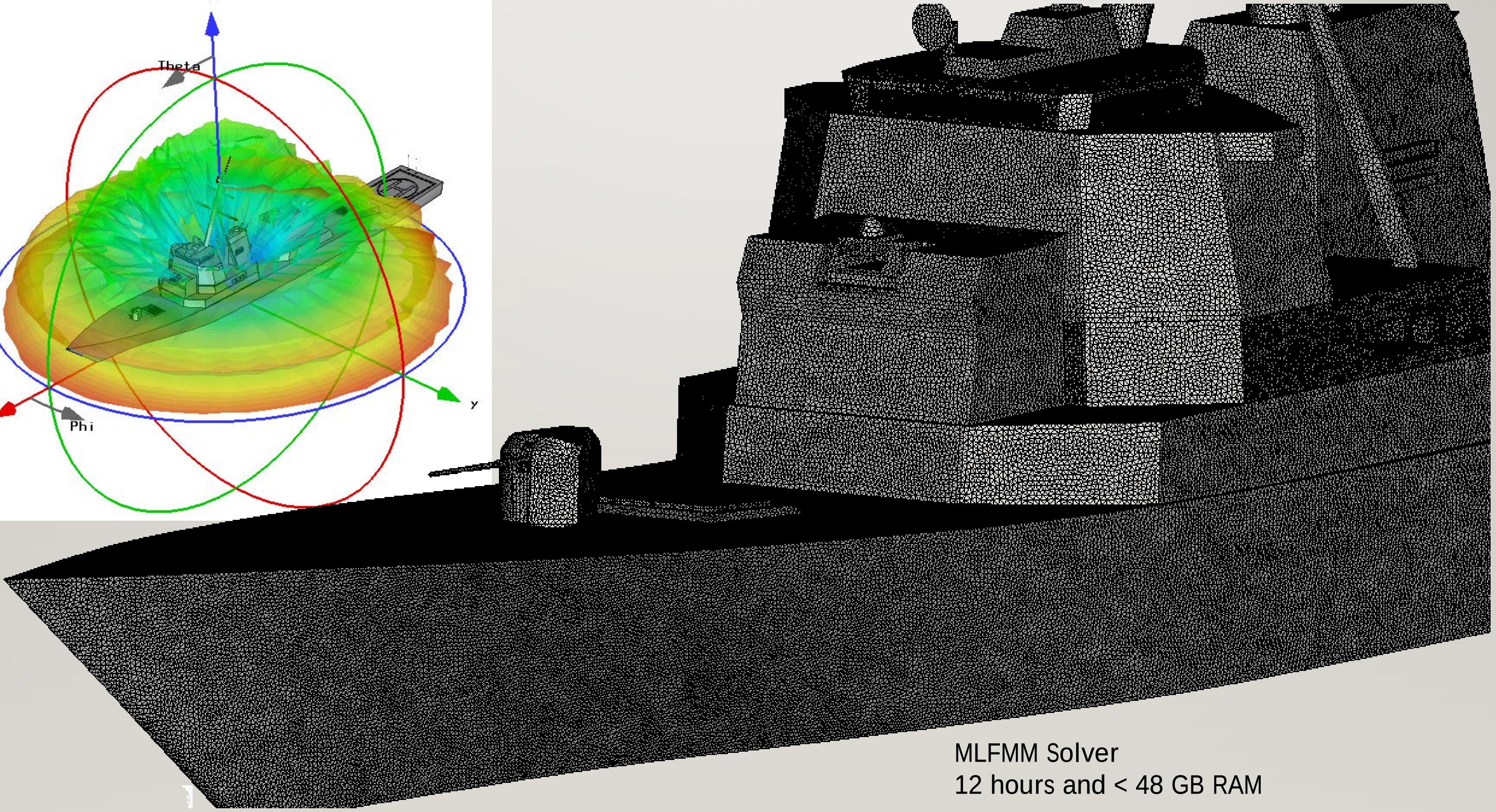
MoM Solver
60 minutes, 12 GB RAM





SHIP SURFACE MESH AT 1 GHZ

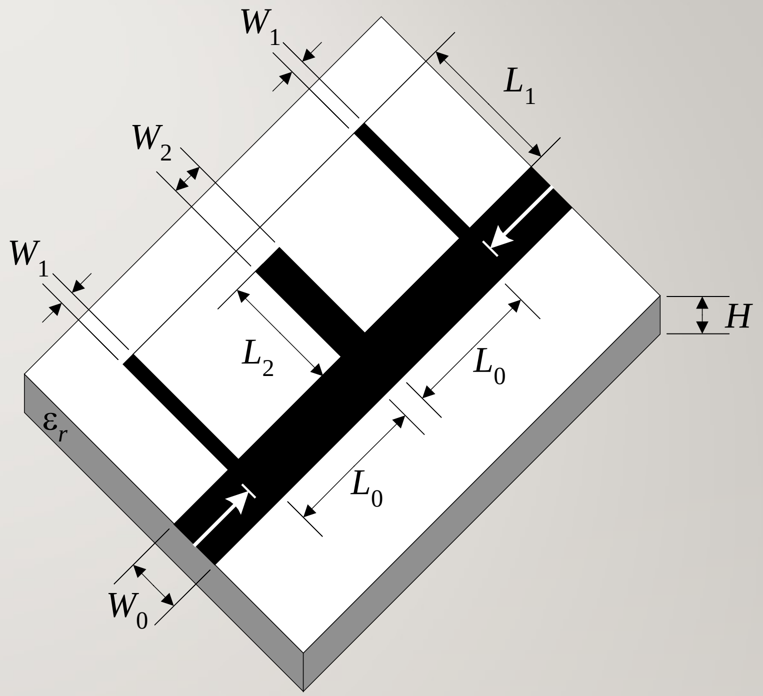




MLFMM Solver
12 hours and < 48 GB RAM

Ship is 500 wavelengths long @ 1 GHz !

Example: Microwave Filter



Task: adjust design variables:
$x = [W_1\ W_2\ L_0\ L_1\ L_2]^T$ so that certain set of specifications is satisfied.

Problems:

1. The filter is evaluated using an electromagnetic simulator Sonnet ***em***. Single evaluation of the objective function takes about **1 hour**.
2. Objective function is only available on **a finite subset** of the domain.
3. There is **no derivative** information available.

Gradient-based search methods do not work in this case!
Grid search methods take too long and often get stuck in local optima!

Surrogate-Based Optimization: Motivation and Concept

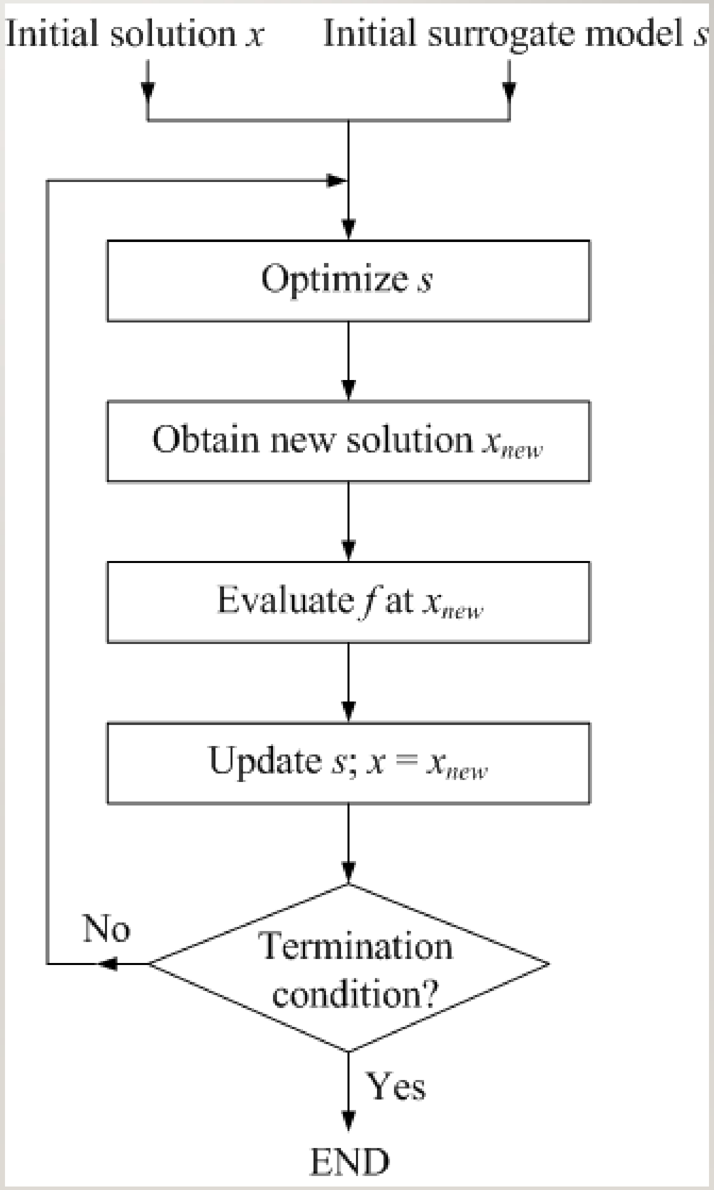
Alternative approach: construct a model *s* of the objective function *f* which:

1. Is computationally cheap
2. Has nice analytical properties
3. Is easy to optimize

Then, iteratively optimize and update the surrogate model *s* using the objective data accumulated during the process until satisfactory solution is found

Surrogate-Based Optimization: Flowchart

1. Construct a surrogate model using a set of known data points.
2. Estimate the function minimizer using the surrogate model.
3. Evaluate the objective function *f* at the estimated minimum.
4. Check for convergence; if achieved, STOP.
5. Update the surrogate model using new data point(s).
6. Go to Step 2


Initial solution x
Initial surrogate model s
Optimize s
Obtain new solution x_{new}
Evaluate f at x_{new}
Update s; $x = x_{new}$
Termination condition?
No
Yes
END

Surrogate-Based Optimization: Flowchart

Important **special case** is a one-shot optimization:

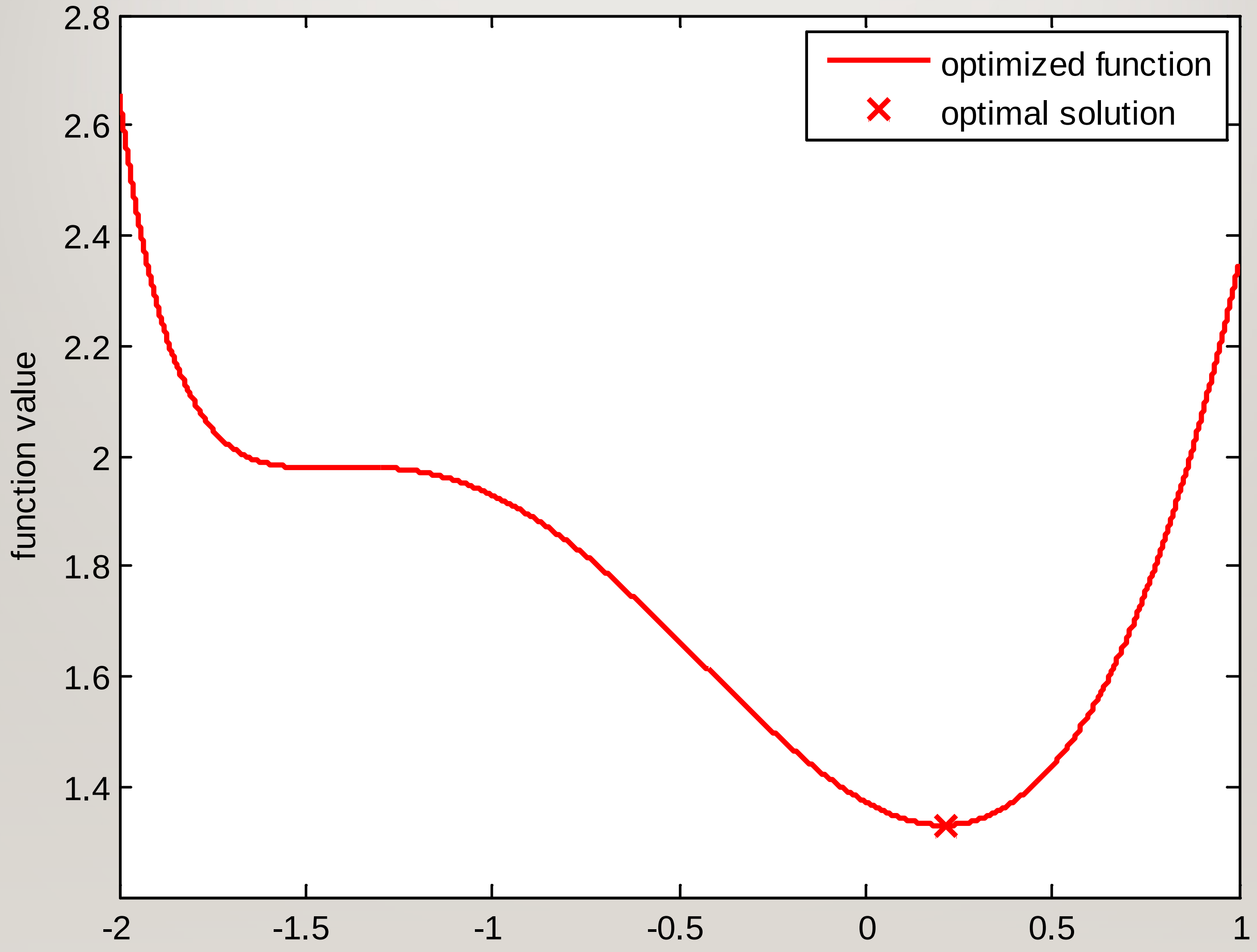
1. The surrogate model *s* of the objective function *f* is created only once using the representative data obtained through appropriate sampling of the search space.
2. The surrogate model is optimized, which concludes the process.

Surrogate-Based Optimization: Flowchart

Surrogate model is also used instead of the original function *f* to perform different design-related tasks such as:

1. Design optimization
2. MO optimization
3. Sensitivity analysis
4. Tolerance analysis
5. Yield-driven design

Surrogate-Based Optimization: Example



Surrogate-Based Optimization: Example

Goal: find minimum in the interval [-2,1]

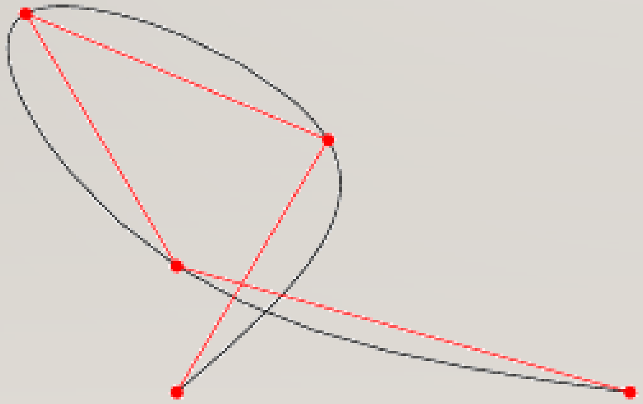
Surrogate model: **cubic splines**

Starting point: -0.5

Additional point: -0.4 (in order to set up non-trivial initial surrogate)

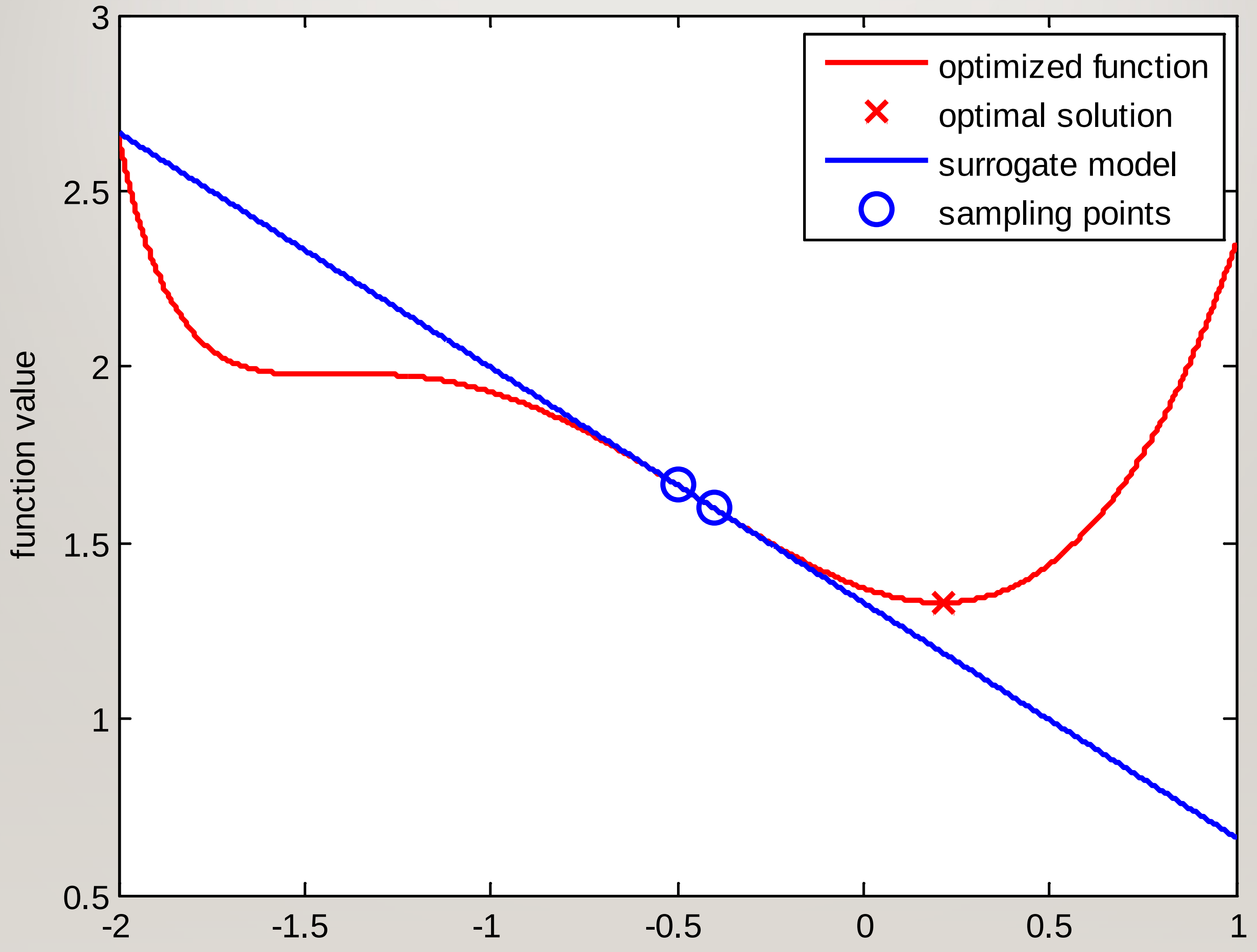
Cubic Spline

a spline constructed of piecewise **third-order polynomials** which pass through a set of m control points.
The **second derivative** of each polynomial is commonly set to **zero** at the endpoints, since this provides a boundary condition that completes the system of equations.
This produces a so-called "natural" cubic spline and leads to a simple tridiagonal system which can be solved easily to give the coefficients of the polynomials. However, this choice is not the only one possible, and other boundary conditions can be used instead.

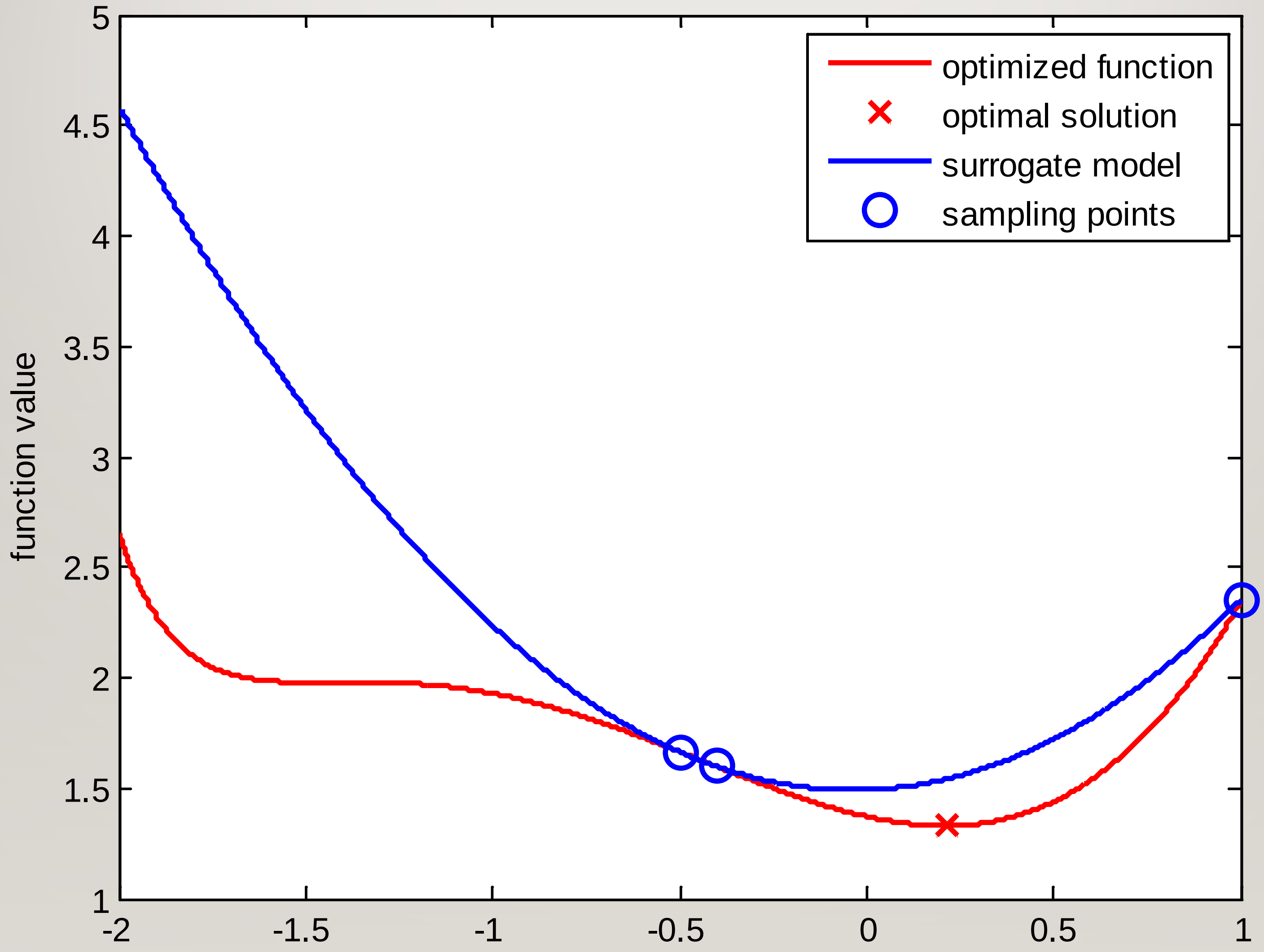


$$\begin{bmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_{n-1} \\ D_n \end{bmatrix} = \begin{bmatrix} 3(y_1 - y_0) \\ 3(y_2 - y_0) \\ 3(y_3 - y_1) \\ \vdots \\ 3(y_{n-1} - y_{n-3}) \\ 3(y_n - y_{n-2}) \\ 3(y_n - y_{n-1}) \end{bmatrix}.$$

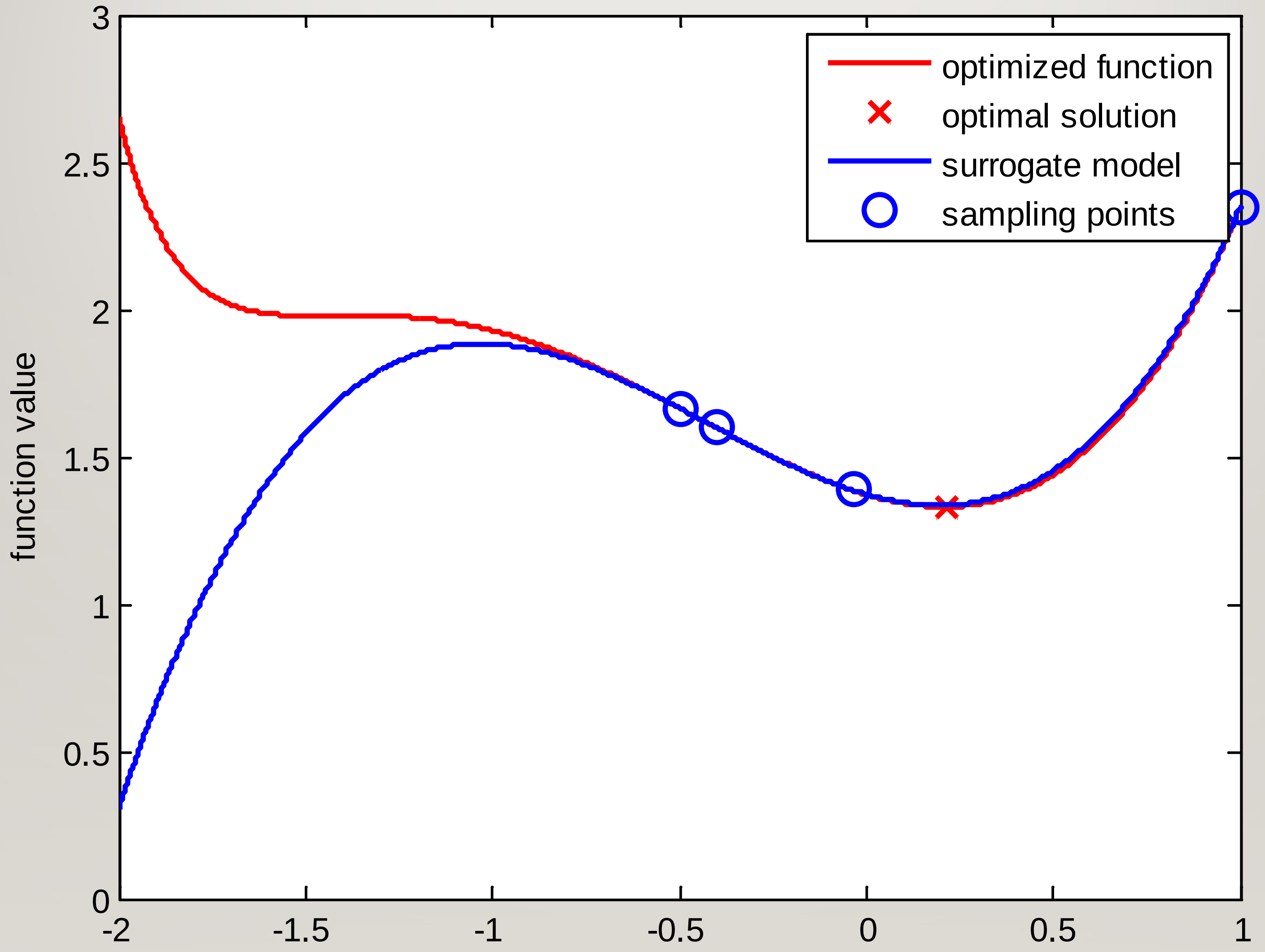
Surrogate-Based Optimization: Example (Iteration 1)



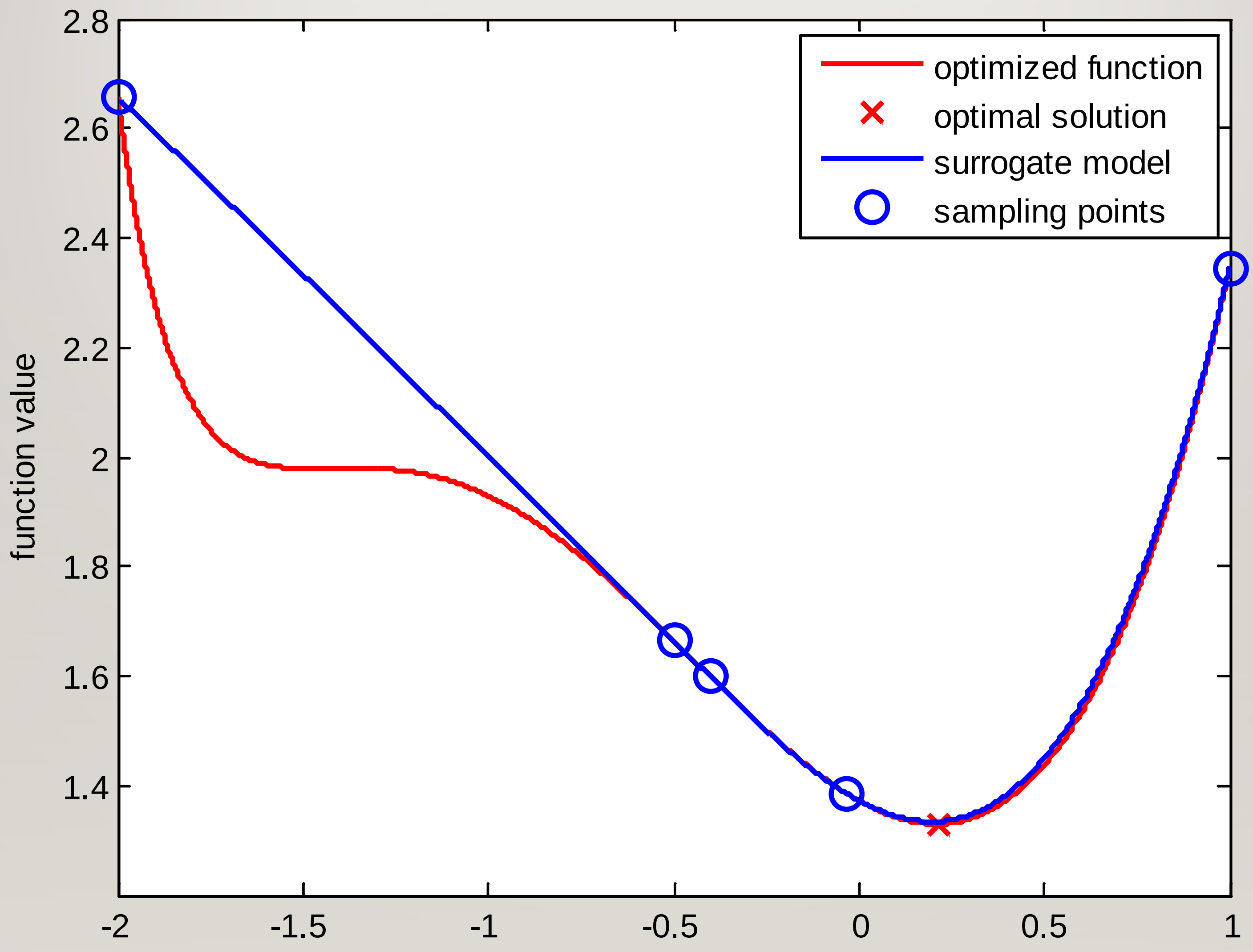
Surrogate-Based Optimization: Example (Iteration 2)



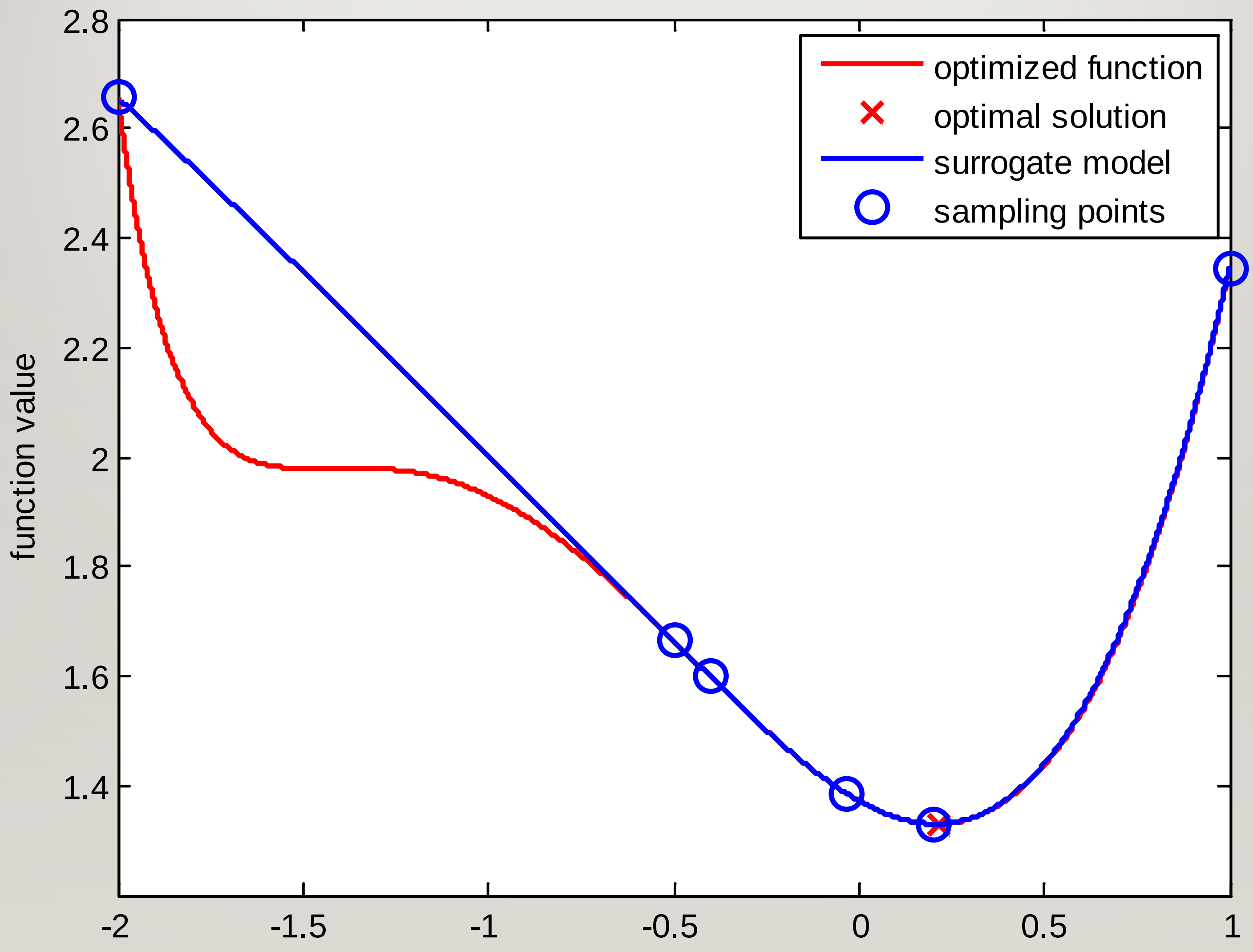
Surrogate-Based Optimization: Example (Iteration 3)



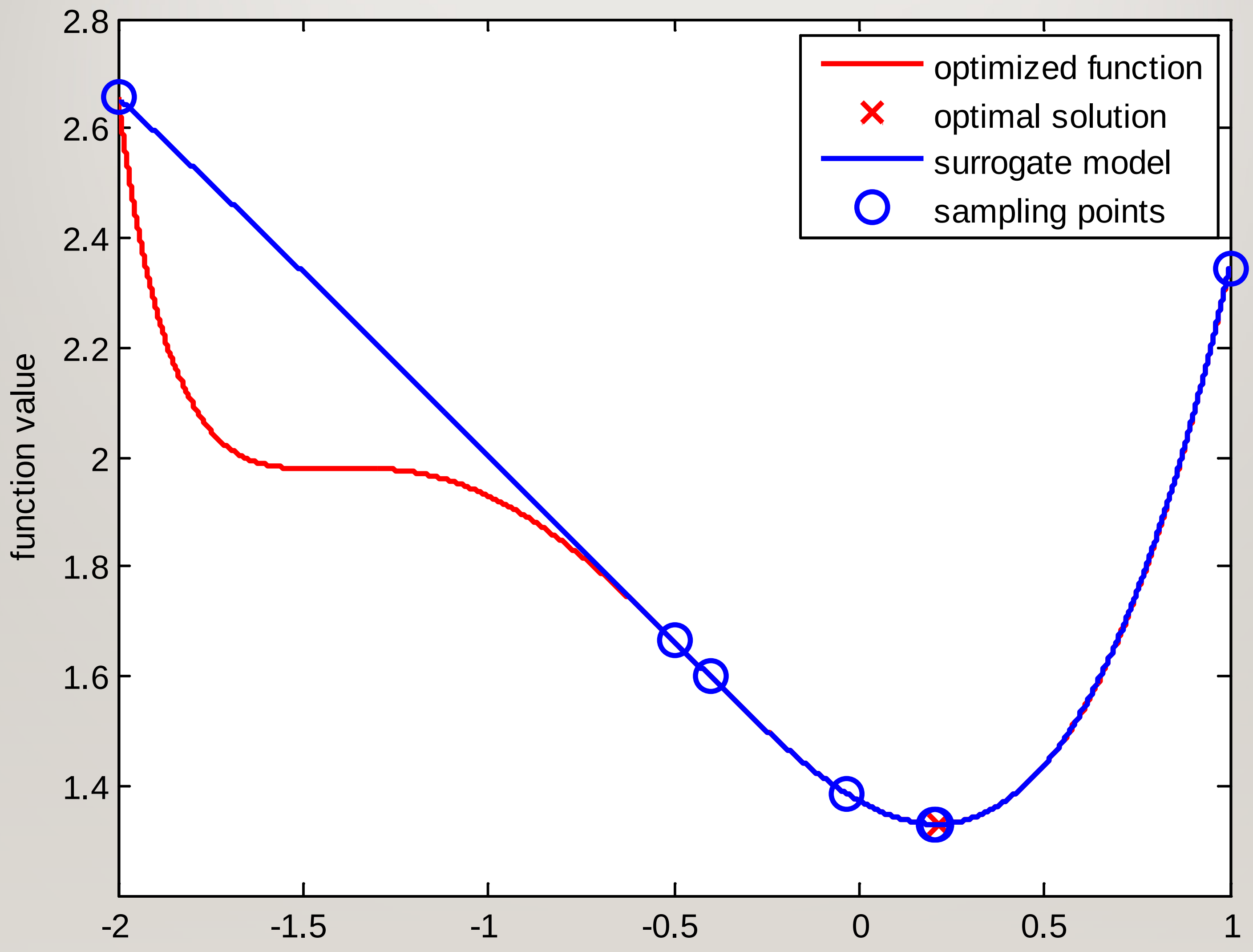
Surrogate-Based Optimization: Example (Iteration 4)



Surrogate-Based Optimization: Example (Iteration 5)



Surrogate-Based Optimization: Example (Iteration 6)



What Can We Infer from the Example?

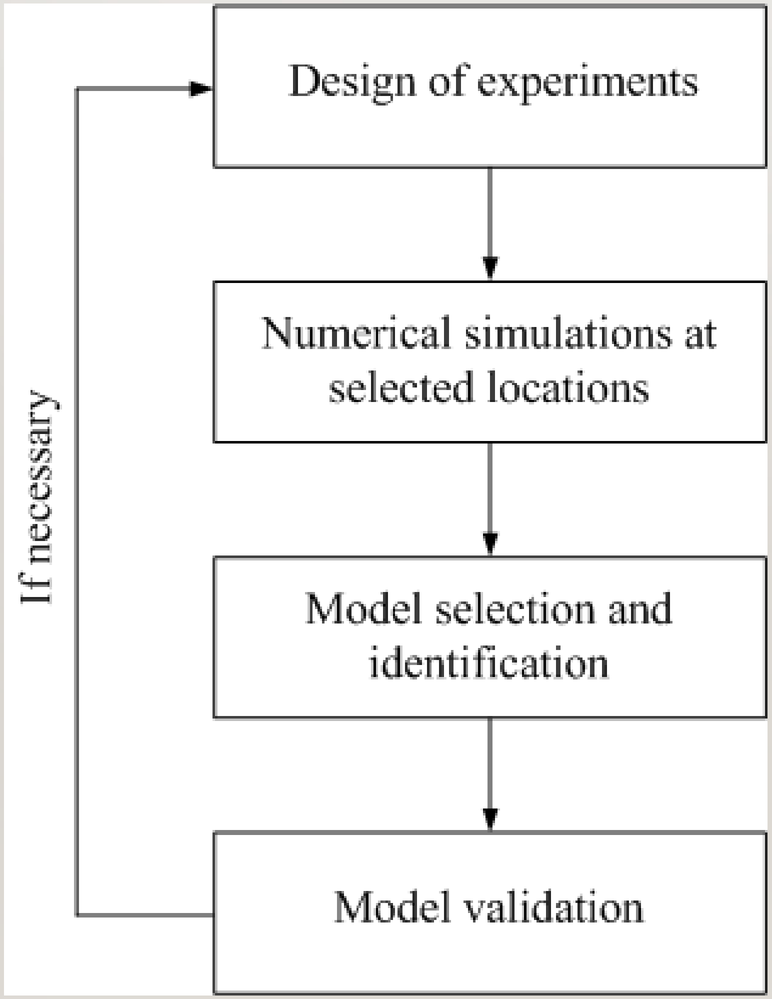
Solution is obtained after **a few evaluations** of the expensive model

Surrogate model undergoes direct optimization and therefore it is expected to be computationally **much cheaper** than the original model

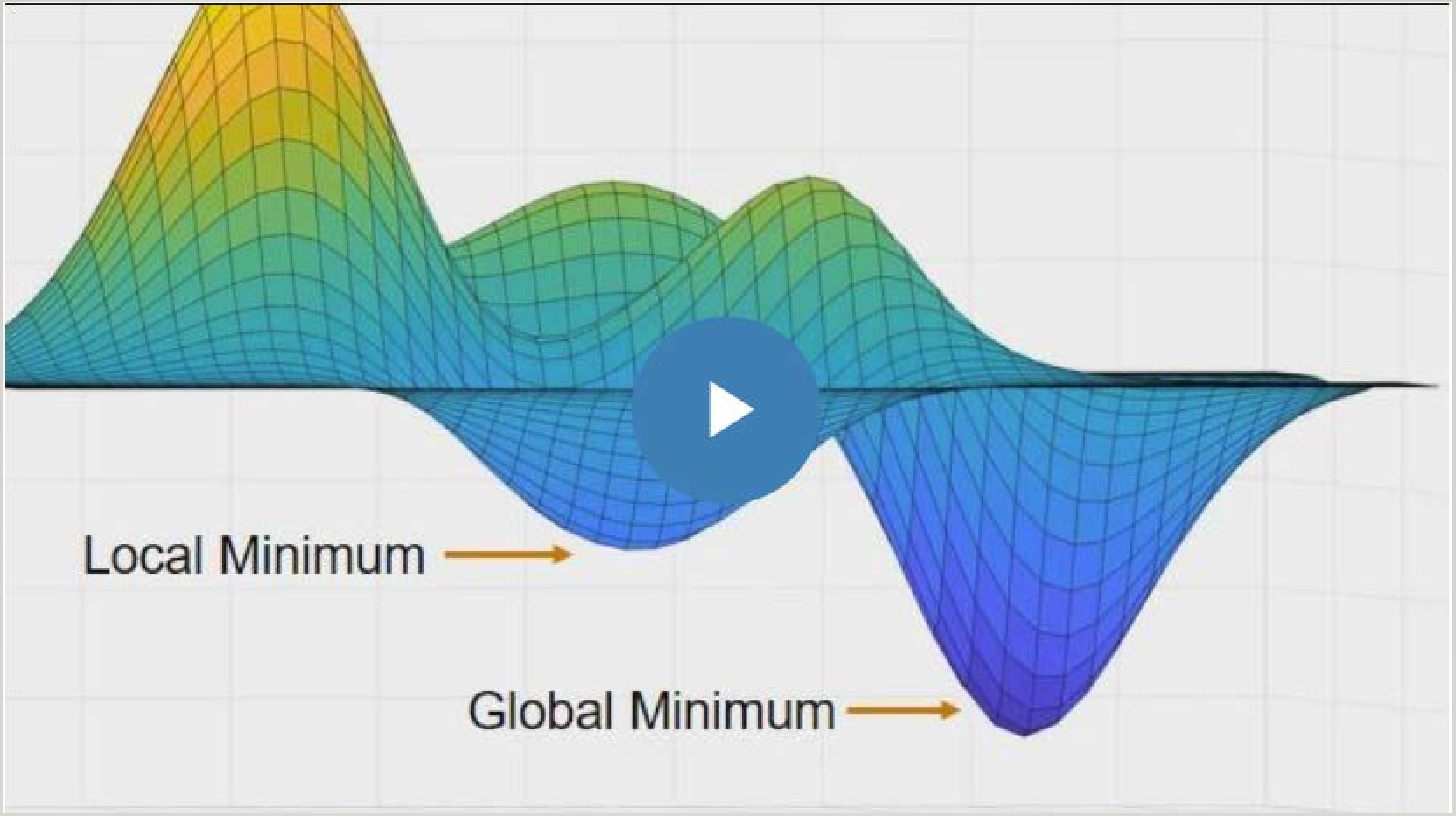
Surrogate model should be good **local representation** of the expensive model; global match is not required in general

Surrogate Modeling: Flowchart

The key steps of the surrogate-based modeling:


Design of experiments
Numerical simulations at selected locations
Model selection and identification
Model validation
If necessary

Surrogate Modeling: Matlab



Surrogate Modeling: Flowchart

<u>Design of experiments (DOE)</u>: the design of experiments is a sampling plan in the design variable space. The important issue is how to assess the goodness of such design, having in mind that the number of samples is normally limited by the computational expense of each sample.

<u>Numerical simulation at selected locations</u>: the computationally expensive model is executed for all design variable vectors specified in the previous step.

Surrogate Modeling: Flowchart

Construction of the surrogate model: two issues are to be answered:

1. What surrogate model should be used (model selection).
2. How do we find the corresponding parameters (model identification)

Model validation: the predictive capabilities of the surrogate model away from the available data (generalization error) should are to be assessed at this step.

Design of Experiments (DOE)

a sampling plan in design variable space;
the function of interest is then evaluated at sampled points and the data pairs are used to create a surrogate model

Typically, the number of points in the data set is severely limited, mainly because of computational expense

Design of Experiments (DOE)

Types of experimental designs:

1. Factorial designs

2. “Space filling” designs

3. Adaptive design

Factorial Designs

Used to estimate main effects, interactions between design variables, and, sometimes, quadratic effects

Typically, the amount of data required for the design grows exponentially with the number of design variables

There is a trade-off between the amount of data used (cost) and the number of effects being estimated

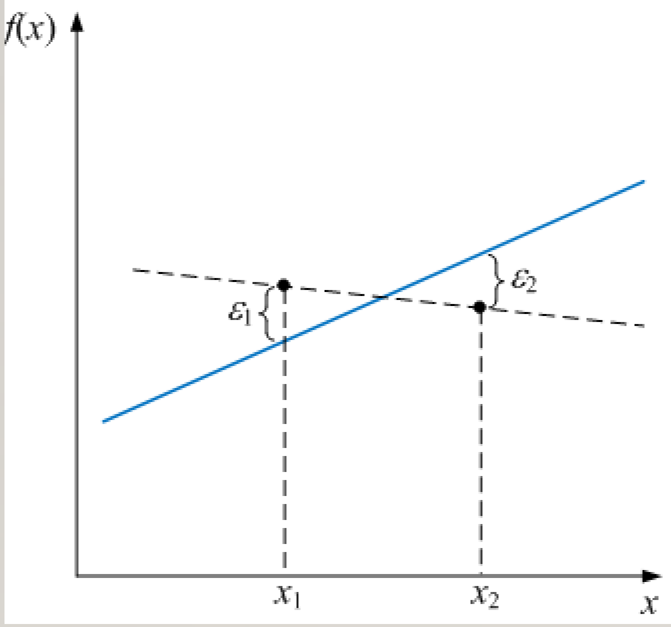
Factorial Designs

Factorial designs are classical methods that assume some randomness and non-repeatability of experiments

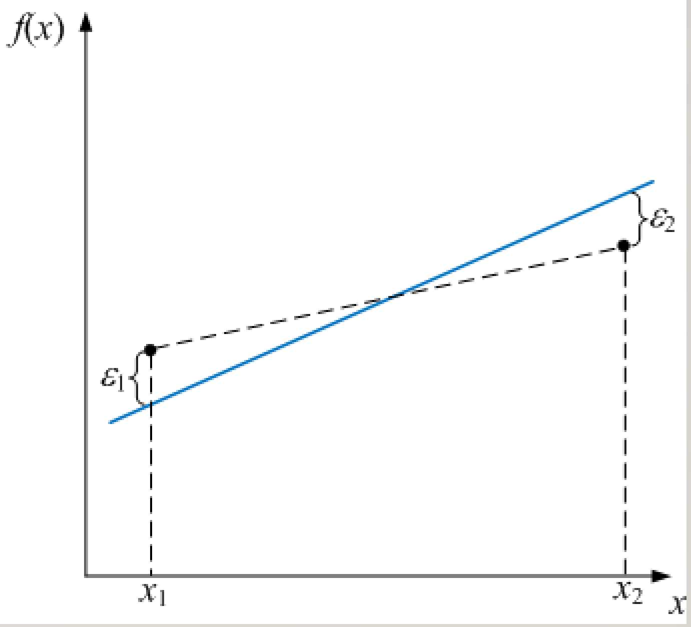
In this context, it is better to **locate samples as far apart as possible** in order to discriminate trend component from the random error component

Factorial Designs

Illustration of the effect of random errors in producing an estimated linear model (dashed line) of the true linear model (solid line):

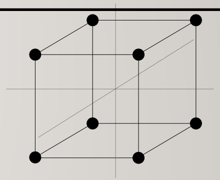
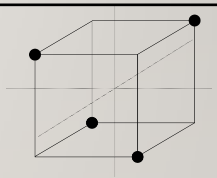
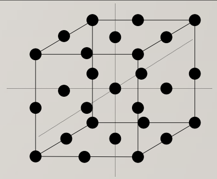
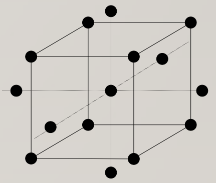
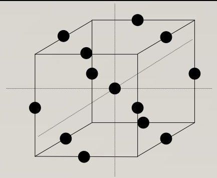
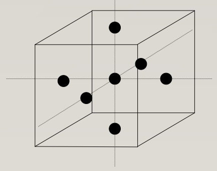


Samples close to each other



Samples on the boundaries of the design space

Factorial Designs: Examples

Type	# of Points	Effects Estimated	3D Example
Full factorial design	2^n	main effects and factor interactions	
Fractional factorial design	2^{n-p} typically $p = 1$	some of main effects and factor interactions	
Block design	3^n	main effects, factor interactions and quadratic effects	
Central composite design	2^n+2n+1 or $2^{n-p}+2n+1$	main effects, factor interactions and some of quadratic effects	
Box-Behnken design	$n \cdot 2^{n-1}+1$	some of main effects, factor interactions and quadratic effects	
Star-distribution design	$2n+1$	main and some of quadratic effects	

“Space Filling” Designs

Supposed to treat all regions of the design space equally

The main requirement is uniformity of the design (e.g., by maximizing the minimum distances among design points, minimizing correlation measures among the sample data, etc.)

The amount of data required for the design is controlled by the user although it grows exponentially with the number of design variables if the design “density” is supposed to be at a “decent” level

Space filling designs are modern methods: non-repeatability component can be omitted since deterministic computer simulations are involved

“Space Filling” Designs: Techniques

1. Random sampling (Monte Carlo)

2. Stratified random sampling

3. Latin hypercube sampling (LHS)

4. Hammersley sequence sampling (HSS)

5. Orthogonal arrays (OA)

6. OA-based LHS

7. Optimal OA-based LHS

$\left.\begin{matrix}5\\6\\7\end{matrix}\right\}$ Not covered in this lecture

“Space Filling” Designs: Monte Carlo Method

Monte Carlo methods vary, but tend to follow a particular pattern:

1. Define a domain of possible inputs
2. Generate inputs randomly from a probability distribution over the domain
3. Perform a deterministic computation on the inputs
4. Aggregate the results

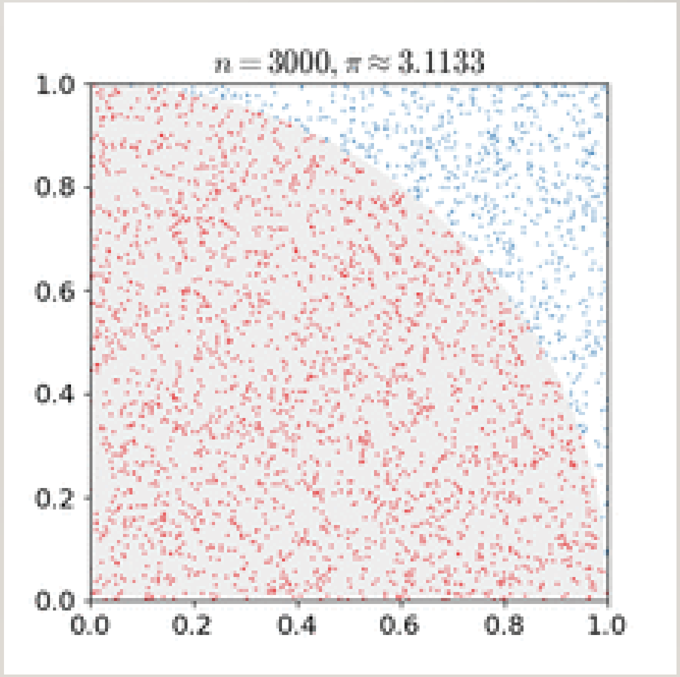
“Space Filling” Designs: Monte Carlo Method

Consider a circle inscribed in a unit square. Given that the circle and the square have a ratio of areas that is π/4, the value of π can be approximated using a Monte Carlo method:

1. Draw a square, then inscribe a circle within it
2. Uniformly scatter objects of uniform size over the square
3. Count the number of objects inside the circle and the total number of objects
4. The ratio of the inside-count and the total-sample-count is an estimate of the ratio of the two areas, which is π/4. Multiply the result by 4 to estimate π.

$$\frac{\pi}{4} \approx \frac{N_{inner}}{N_{total}}$$

$$\pi \approx 4\frac{N_{inner}}{N_{total}}$$



“Space Filling” Designs: Random Sampling

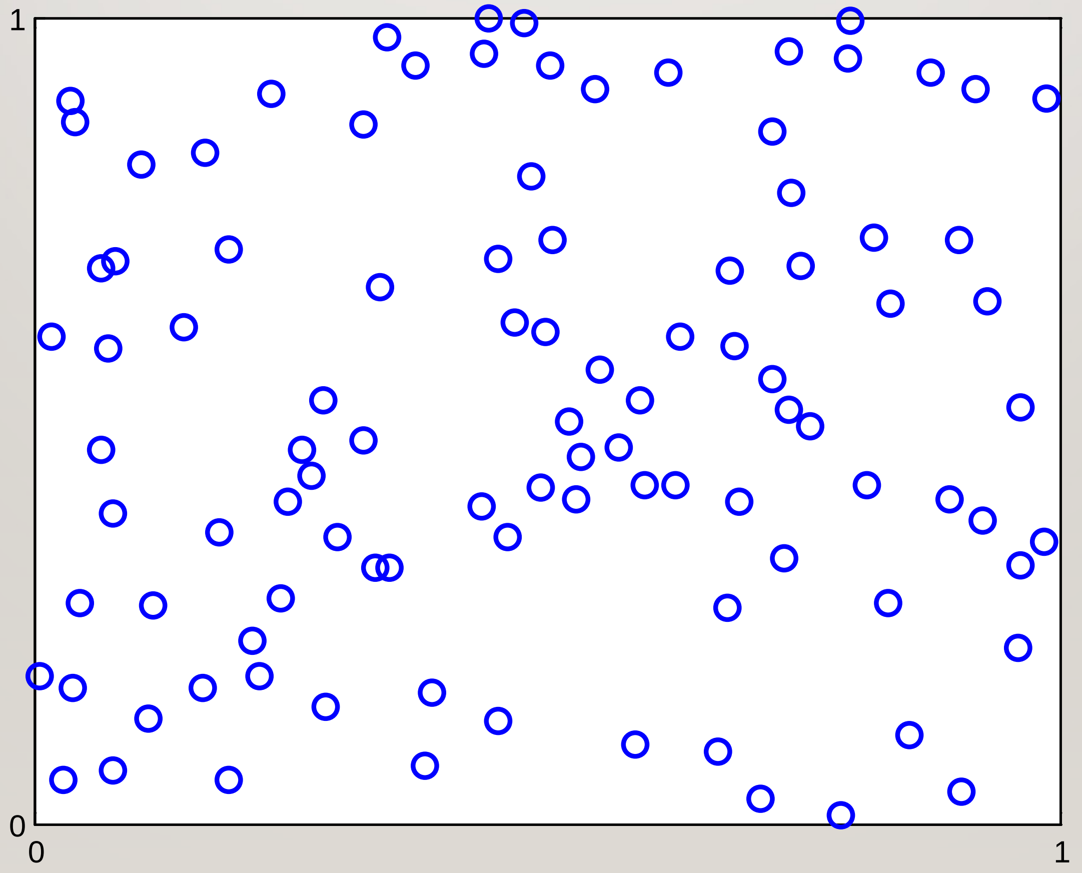
Implementation: assign samples within a design space using pseudo-random number generation algorithm.

Features:

1. Simple.
2. Does not produce really uniform sampling.
3. Typically leaves large regions of the design space unexplored, especially in high-dimensional spaces.

“Space Filling” Designs: Random Sampling

Random sampling example for $n = 2$ and $N = 100$:



“Space Filling” Designs: Stratified Random Sampling

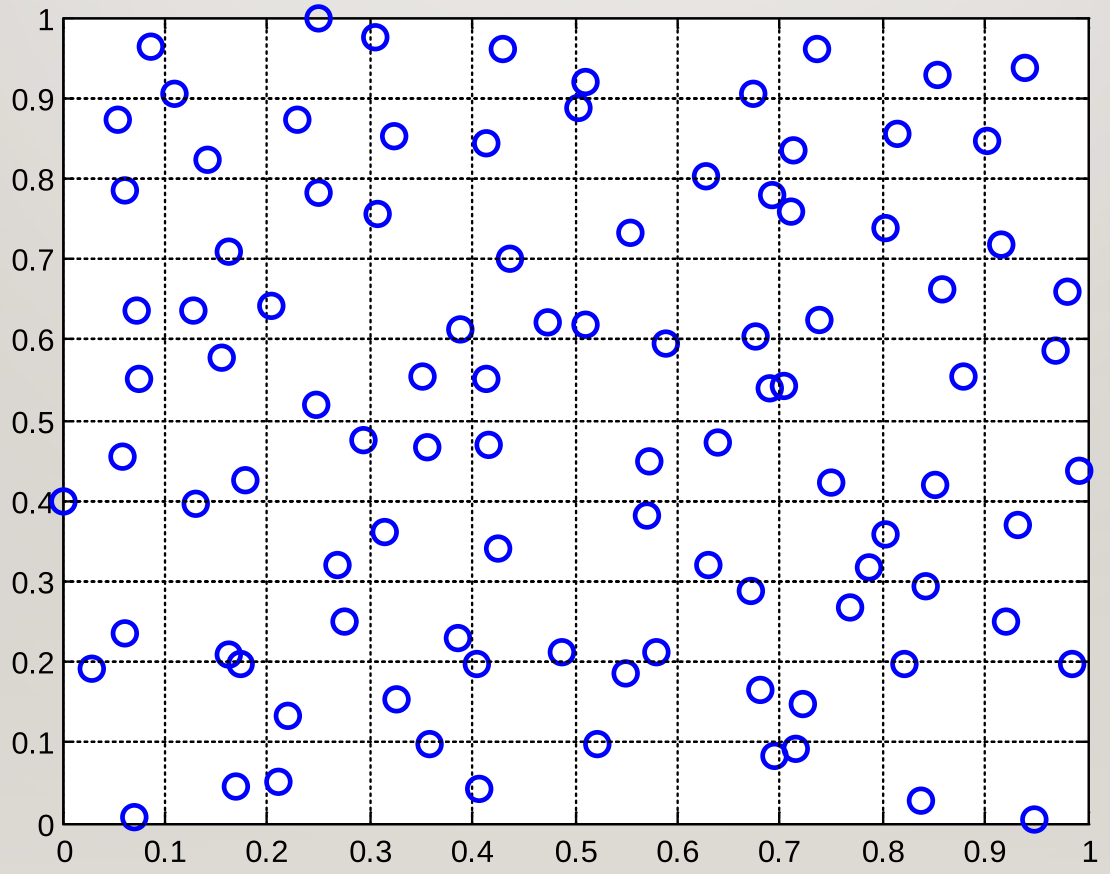
Implementation: divide each of *n* intervals into subintervals and create “bins” of equal probability; put random sample within each bin.

Features:

1. Simple.
2. Produces quite uniform distribution of samples.
3. The number of samples scales at best as 2^n, which may be impractical, especially for large *n*.

“Space Filling” Designs: Stratified Random Sampling

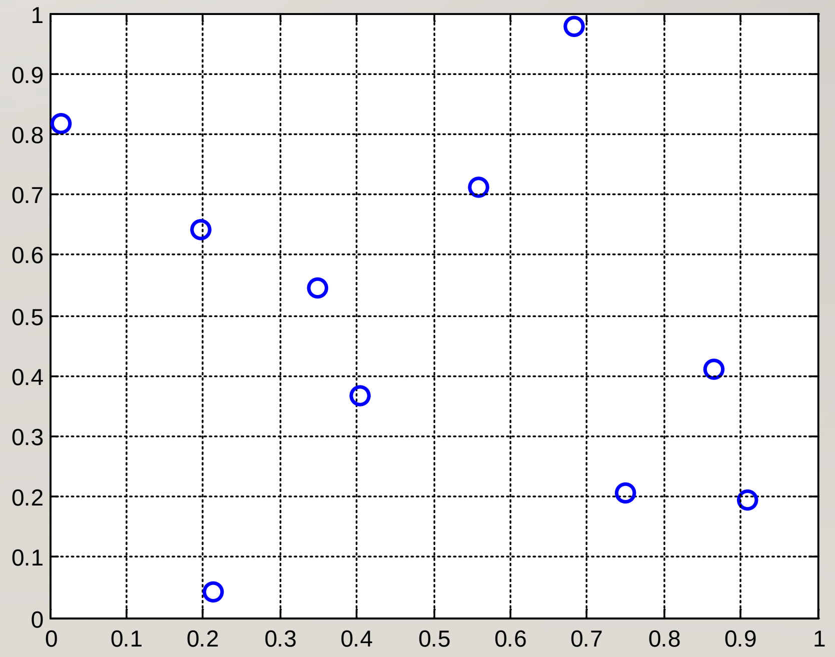
Stratified random sampling example for *n* = 2 and *N* = 100:



“Space Filling” Designs: Latin Hypercube Sampling (LHS)

Implementation: divide each of n intervals into N subintervals, where N is the number of samples (this yields N^n bins in the design space). Next, select N samples so that (i) each sample is randomly placed inside a bin, and (ii) for all one-dimensional projections of the N samples and bins, there will be one and only one sample in each bin.

LHS for $N = 10$ samples for $n = 2$: each interval is divided into N subintervals; samples are placed so that **each row and each column contains exactly one sample**.



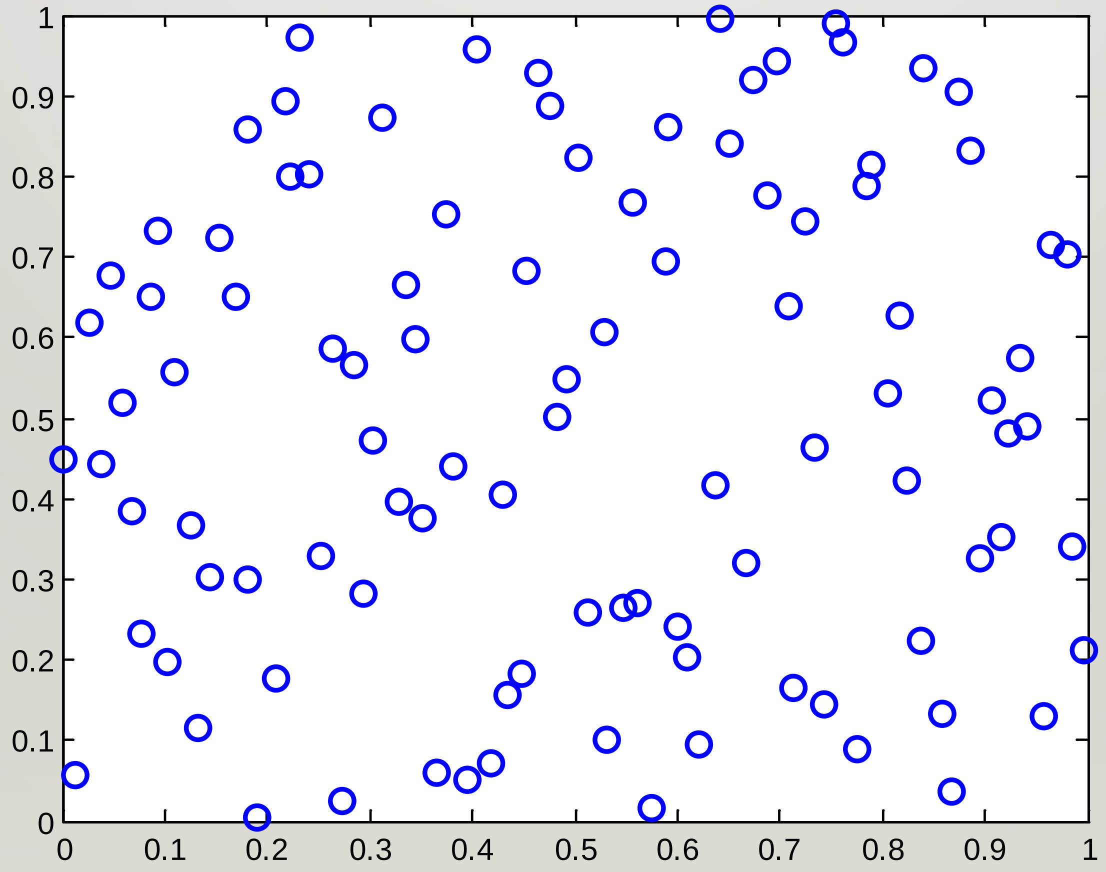
“Space Filling” Designs: Latin Hypercube Sampling

Features:

1. Typically produces better (more uniform) distribution of samples than Monte Carlo method.
2. Can be configured with any number of samples; in particular, it is not restricted to sample sizes that are specific multiplies or powers of *n*.
3. There is more than one possible arrangement of bins and samples that meets the LHS criteria. For example, all samples being nearly co-linear (e.g., placed along one of the diagonals) is a poor arrangement but it still satisfies LHS criteria.
4. There are extensions to the basic LHS technique that minimize the amount of correlation among the samples.

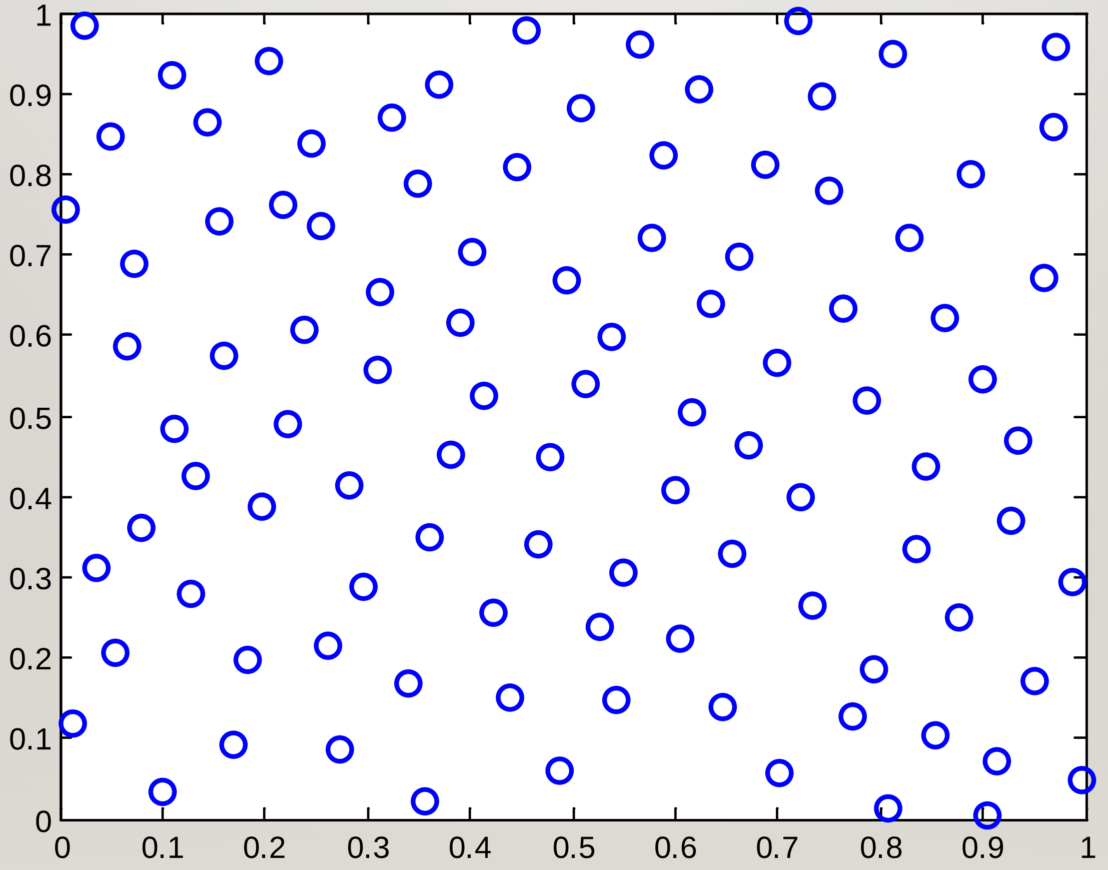
"Space Filling" Designs: Latin Hypercube Sampling

Latin hypercube sampling example for $n = 2$ and $N = 100$:



“Space Filling” Designs: Latin Hypercube Sampling

Improved Latin hypercube sampling example for *n* = 2 and *N* = 100:



“Space Filling” Designs: Hammersley Sampling

Hammersley sampling is a variant of quasi-Monte Carlo sampling method that generates points using the radix-*R* notation of an integer.

A specific integer *p* can be represented in radix-*R* notation (*R* is integer, e.g. 2, 8, 10) as

$$p = p_m p_{m-1} \ldots p_2 p_1 p_0$$

$$p = p_0 + p_1 R + p_2 R^2 + \ldots + p_m R^m$$

Hammersley sampling uses the inverse radix number representation of the form

$$0.p_0 p_1 p_2 \ldots p_m$$

$$p_0 R^{-1} + p_1 R^{-2} + p_2 R^{-3} + \ldots + p_m R^{-m-1}$$

“Space Filling” Designs: Hammersley Sampling

The Hammersley sequence of *n*-dimensional points is generated as

$$x^{(j)} = \begin{bmatrix} \dfrac{j}{N} & \varphi_{R_1}(j) & \varphi_{R_2}(j) & \cdots & \varphi_{R_{n-1}}(j) \end{bmatrix}^T$$

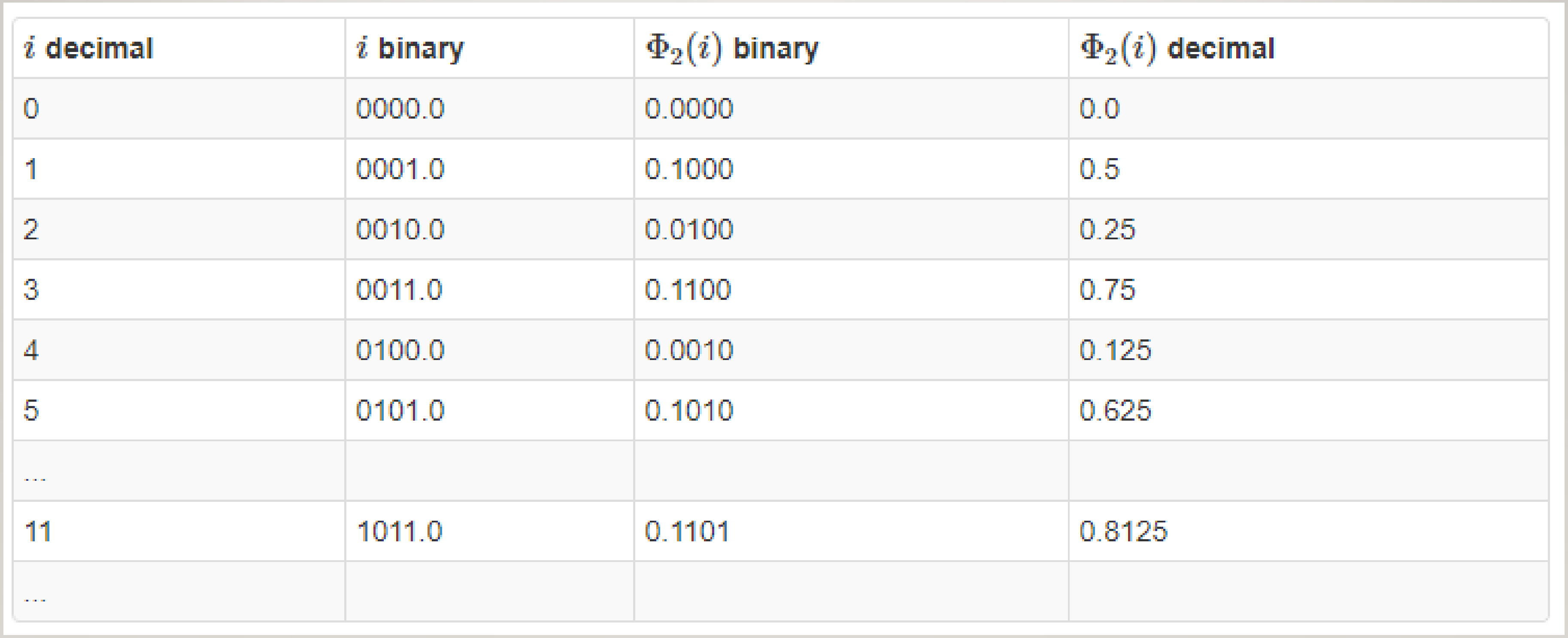
where $j = 0, 1, \ldots, N\text{-}1$, and R_1, R_2, …, are the first *n*-1 prime numbers (2, 3, 5, 7, 11, 13, …), $\varphi_R(p)$ is

$$\varphi_R(p) = p_0 R^{-1} + p_1 R^{-2} + p_2 R^{-3} + \ldots + p_m R^{-m-1}$$

This approach generates a set of *N* points in the *n*-dimensional design space $[0,1]^n$.

“Space Filling” Designs: Hammersley Sampling

Inverse radix number for binary (Radix-2)



i decimal	i binary	$\Phi_2(i)$ binary	$\Phi_2(i)$ decimal
0	0000.0	0.0000	0.0
1	0001.0	0.1000	0.5
2	0010.0	0.0100	0.25
3	0011.0	0.1100	0.75
4	0100.0	0.0010	0.125
5	0101.0	0.1010	0.625
...			
11	1011.0	0.1101	0.8125
...			

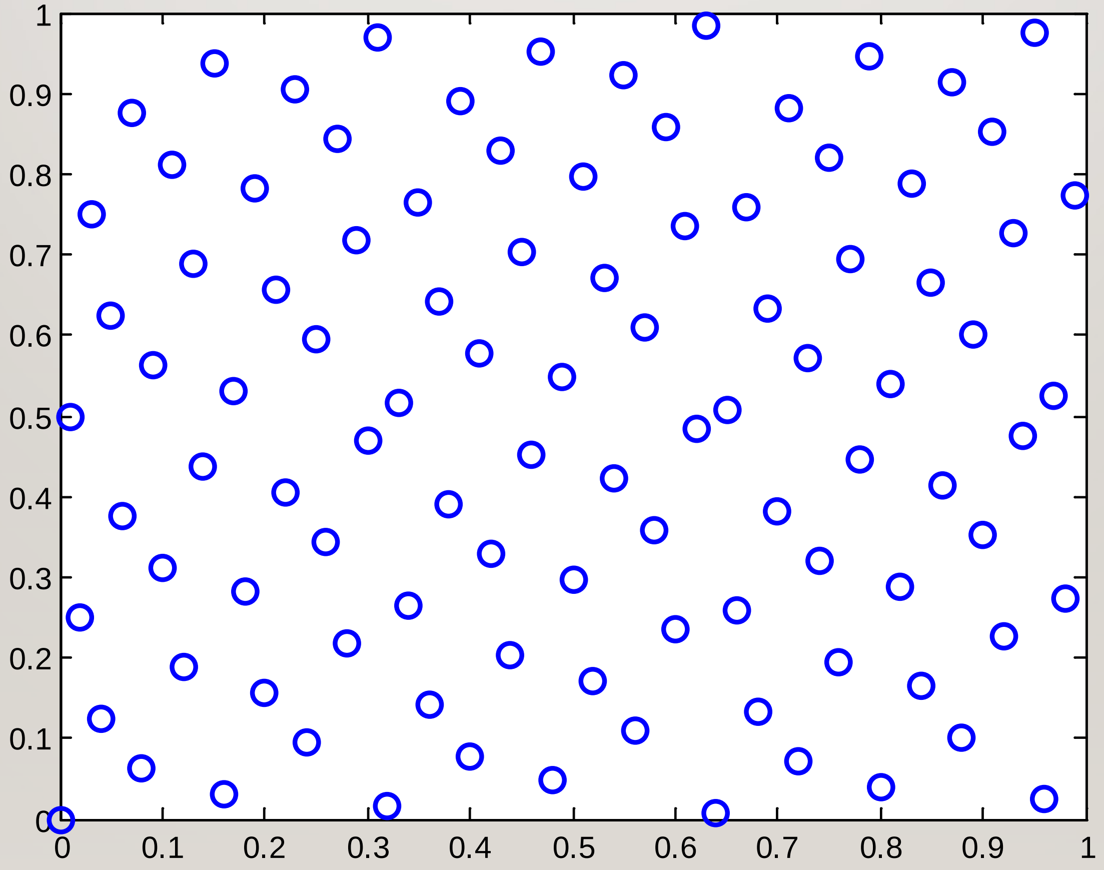
“Space Filling” Designs: Hammersley Sampling

Features:

1. Produces very good (uniform) distribution of samples.
2. Similarly as LHS, Hammersley method produce any number of samples; in particular, it is not restricted to sample sizes that are specific multiplies or powers of *n*.

“Space Filling” Designs: Hammersley Sampling

Hammersley sampling example for n = 2 and N = 100:



point (0,0) is always part of this sequence. You can also see that the points are quite well distributed for any number of points.

“Space Filling” Designs: Optimization-Based Uniform Sampling

It is possible to apply optimization techniques in order to improve uniformity of the distribution of samples in the design space.

Practical uniformity metric:

$$E = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{1}{\| x^{(i)} - x^{(j)} \|^2}$$

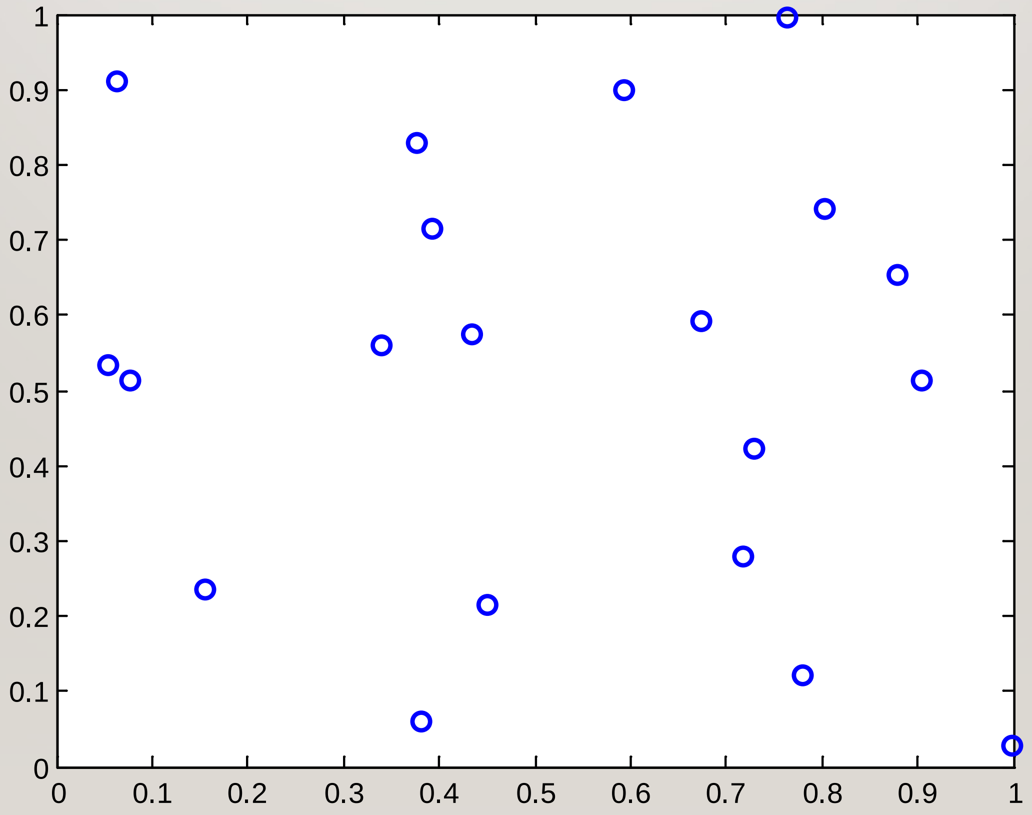
where $x^{(j)}$, $j = 1, \ldots, N$ are samples

Any optimization routine may be applied to minimize E as a function of sample locations.

Remark: this procedure may be computationally expensive, especially for large N.

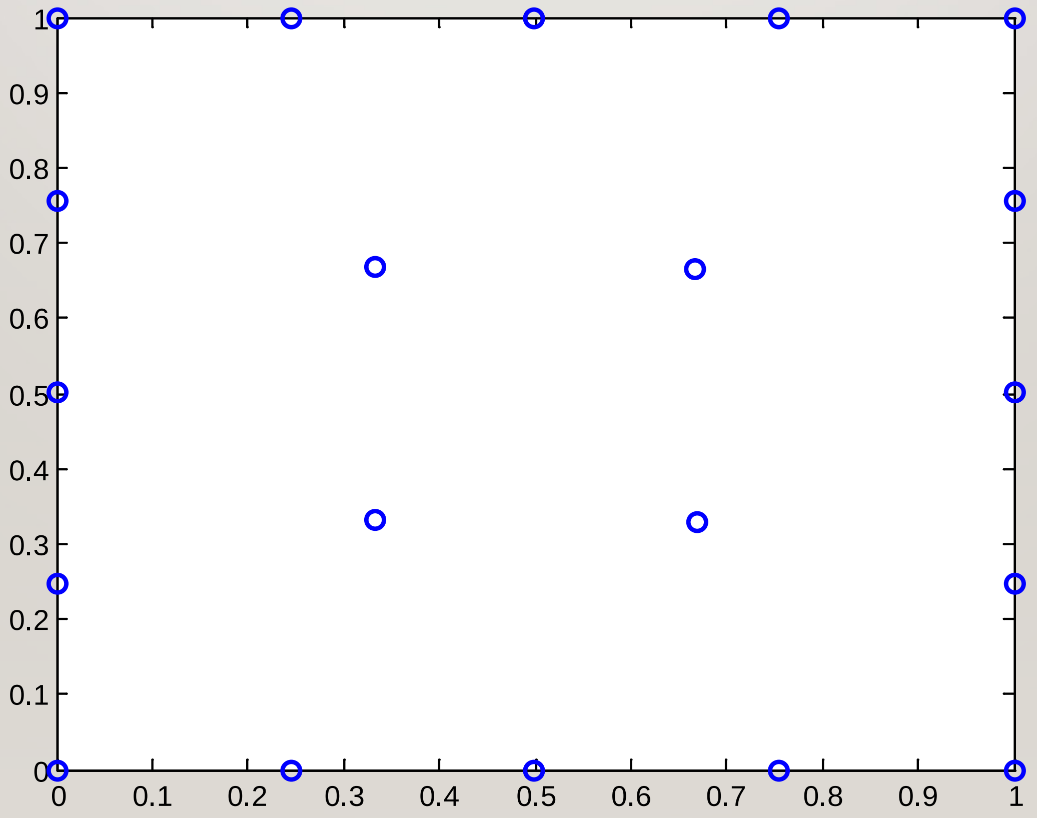
“Space Filling” Designs: Optimization-Based Uniform Sampling

Example: $n = 2$, $N = 20$
Initial distribution (random sampling):



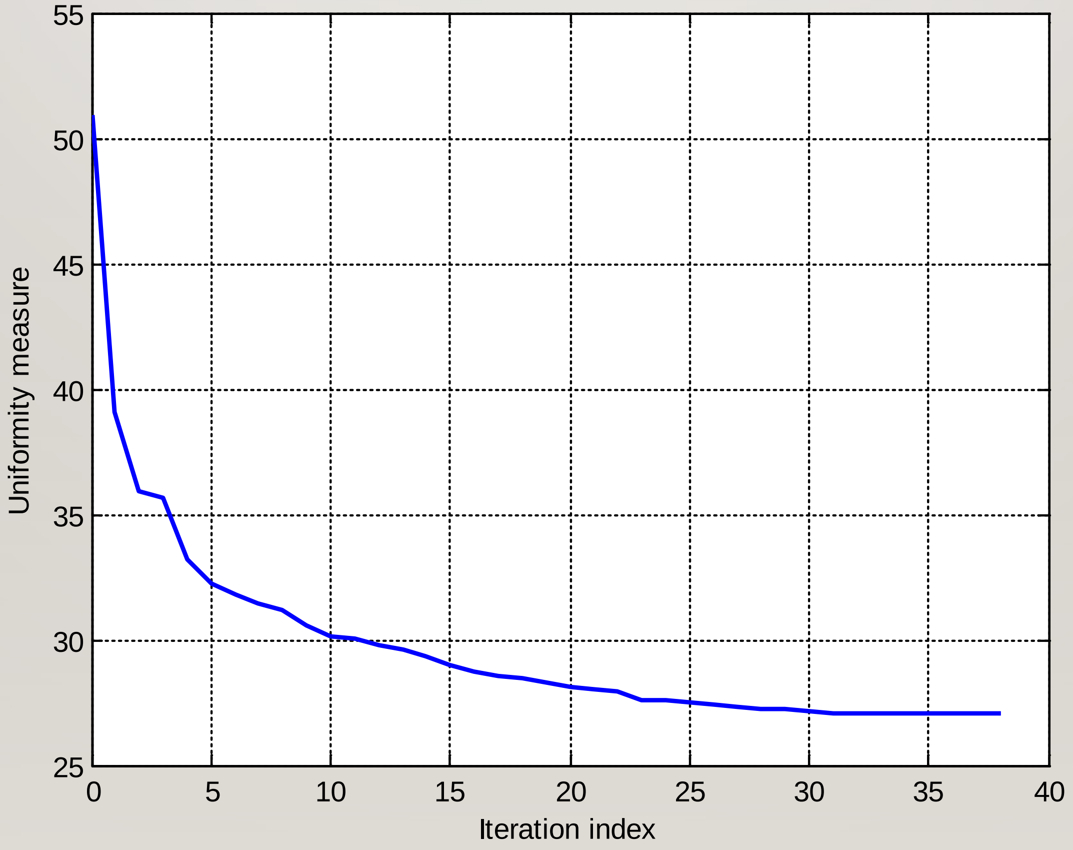
“Space Filling” Designs: Optimization-Based Uniform Sampling

Example: $n = 2$, $N = 20$
Final distribution:



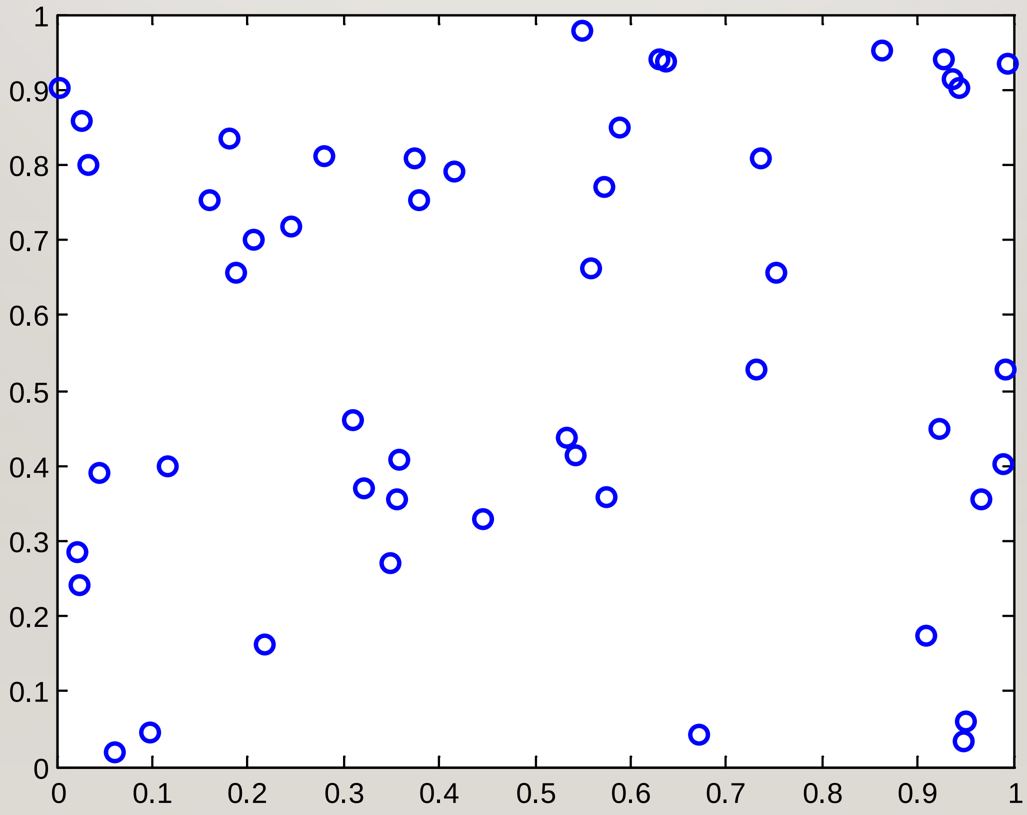
“Space Filling” Designs: Optimization-Based Uniform Sampling

Example: $n = 2$, $N = 20$
Evolution of the uniformity measure:



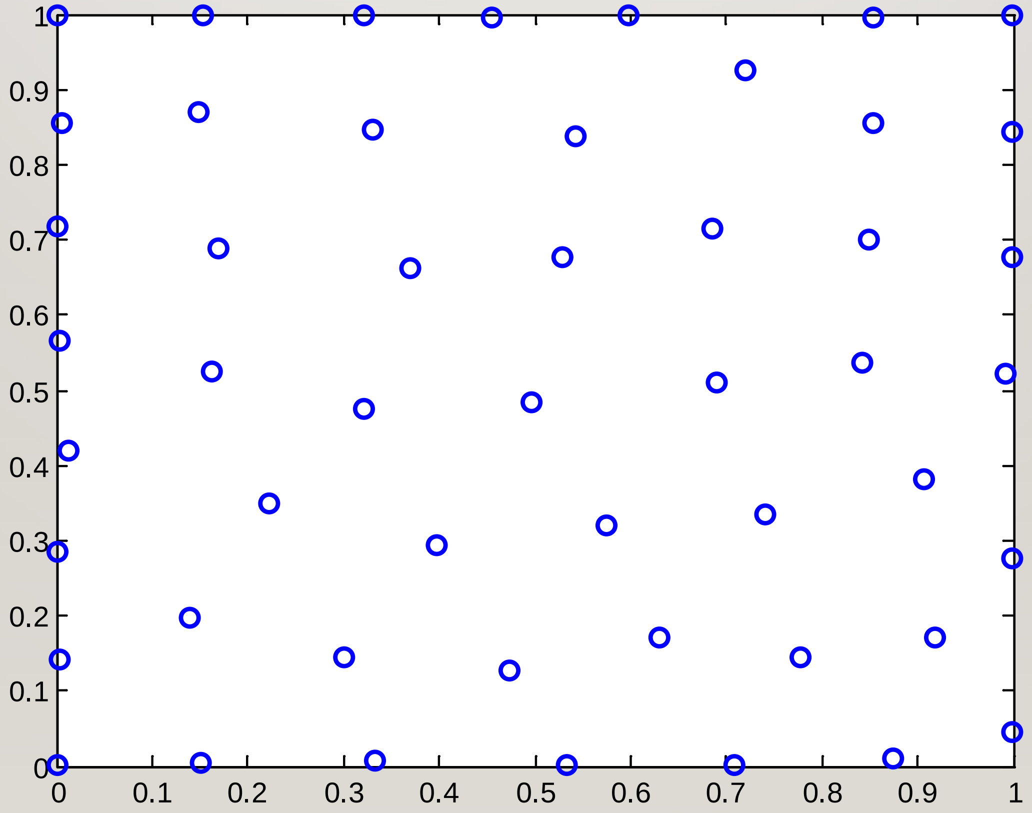
“Space Filling” Designs: Optimization-Based Uniform Sampling

Example: $n = 2$, $N = 50$
Initial distribution (random sampling):



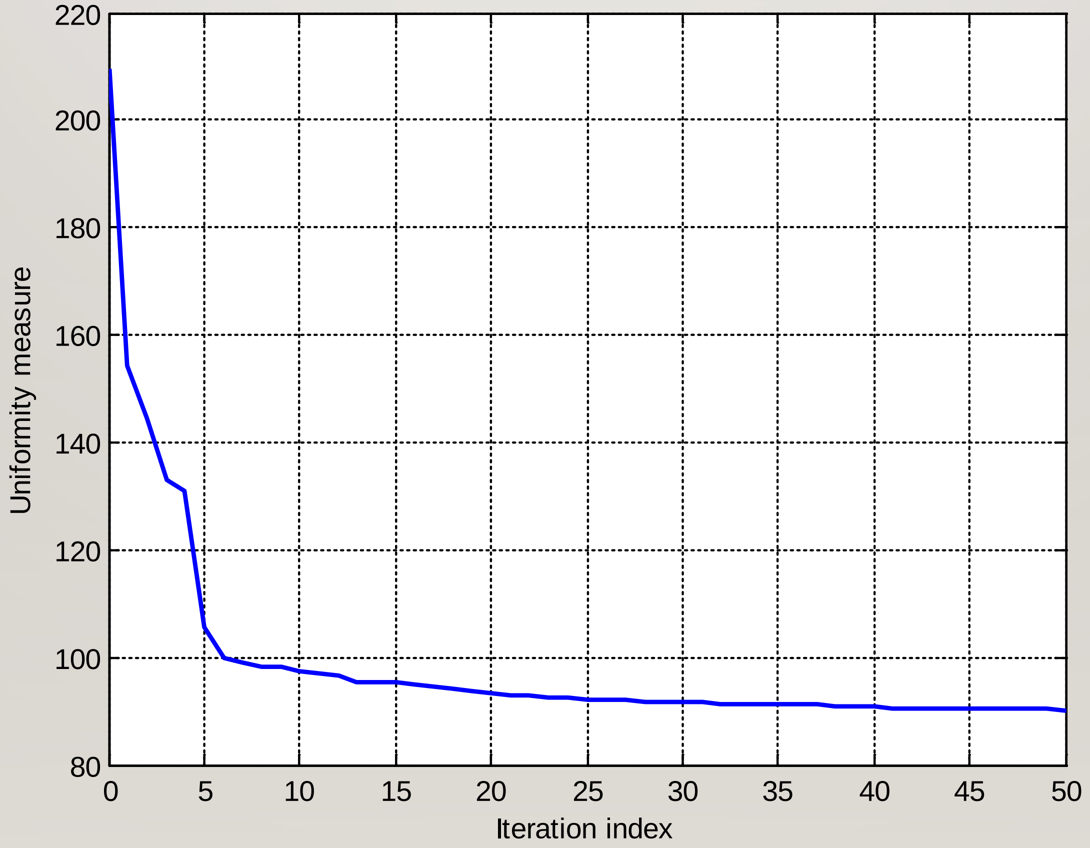
“Space Filling” Designs: Optimization-Based Uniform Sampling

Example: $n = 2$, $N = 50$
Final distribution:



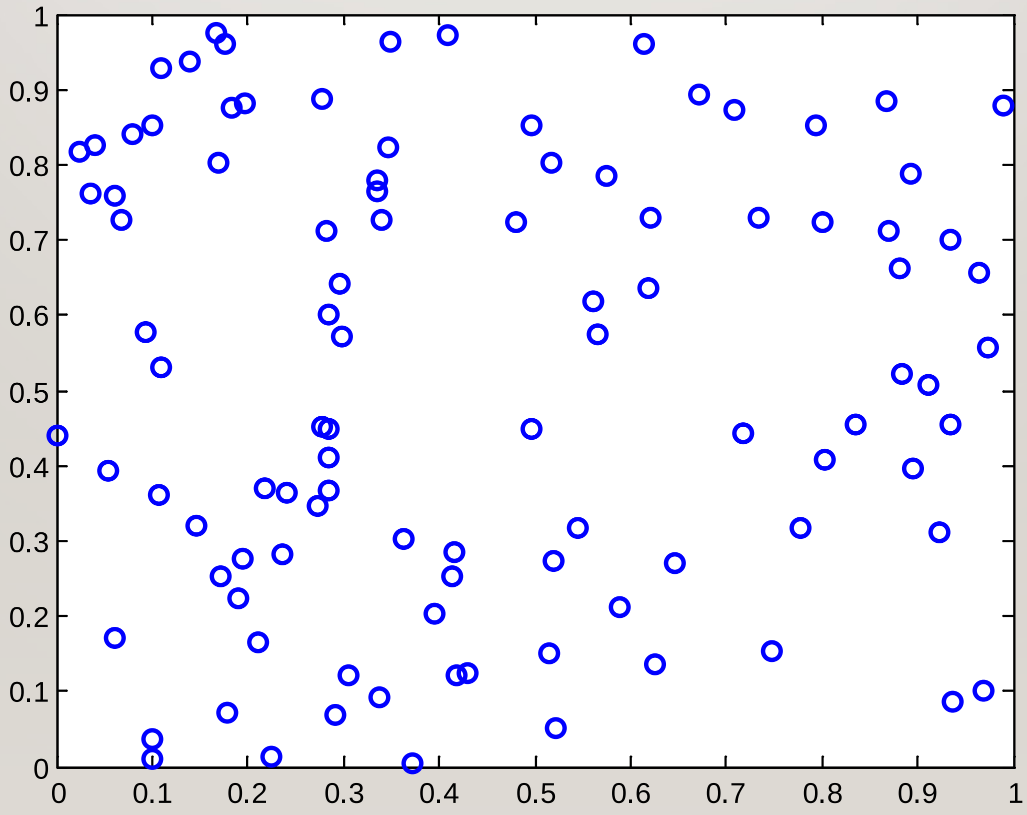
“Space Filling” Designs: Optimization-Based Uniform Sampling

Example: $n = 2$, $N = 50$
Evolution of the uniformity measure:



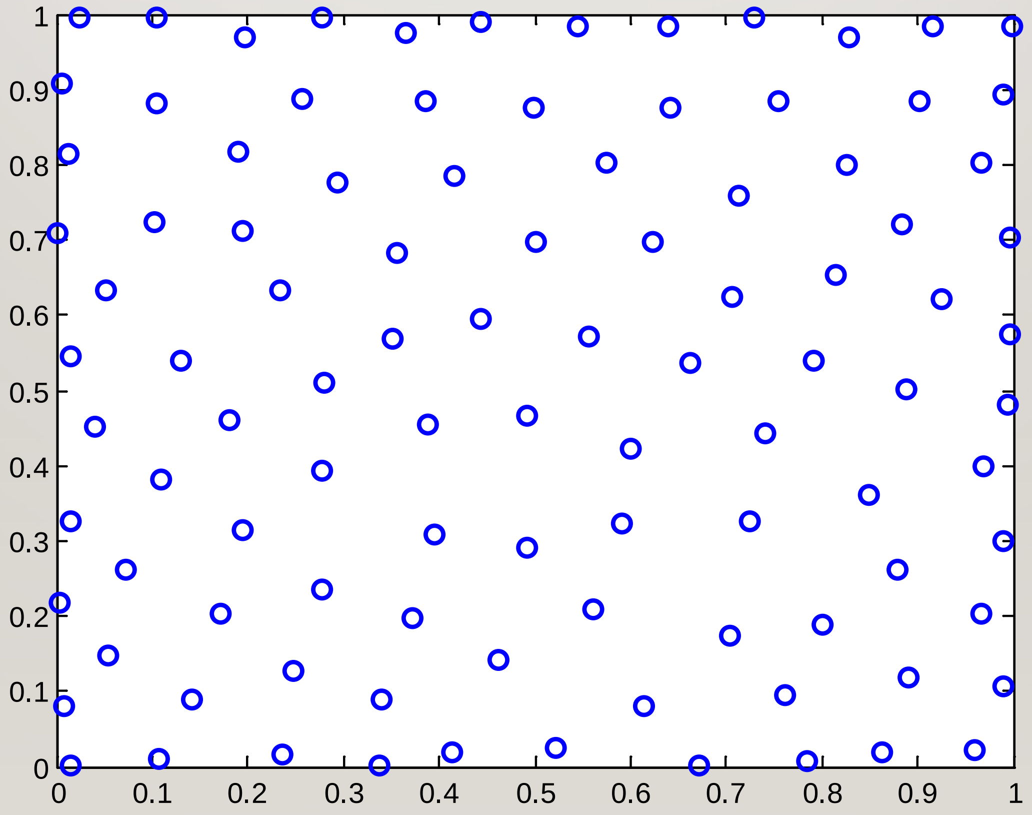
“Space Filling” Designs: Optimization-Based Uniform Sampling

Example: $n = 2$, $N = 100$
Initial distribution (random sampling):



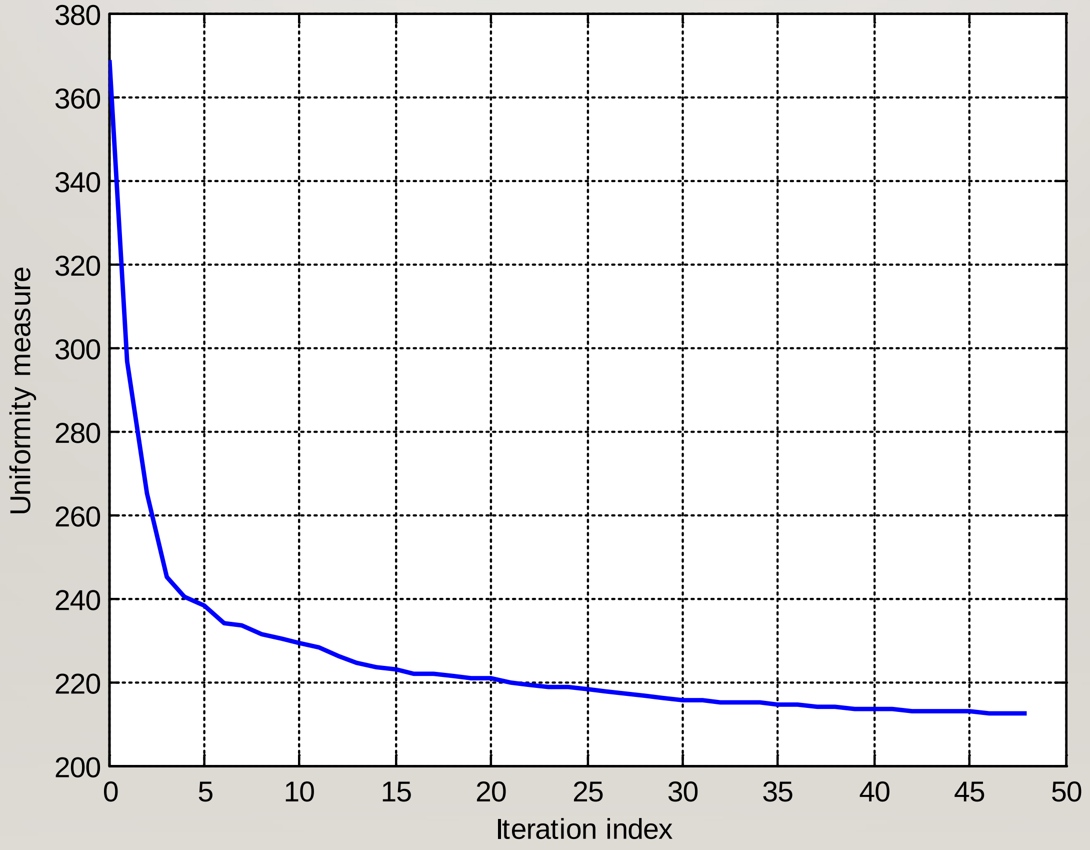
“Space Filling” Designs: Optimization-Based Uniform Sampling

Example: $n = 2$, $N = 100$
Final distribution:



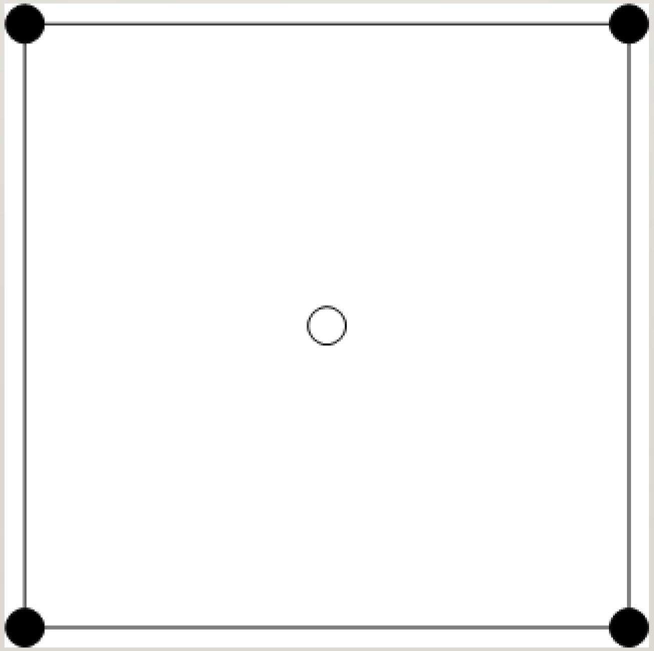
“Space Filling” Designs: Optimization-Based Uniform Sampling

Example: $n = 2$, $N = 100$
Evolution of the uniformity measure:



Adaptive Designs

Adaptive design adjusts the data set by inserting more data samples into regions of largest modeling error
Example: Start with a uniform mesh (2^n samples) and put a validation point in the middle; evaluate the modeling error at the validation point:



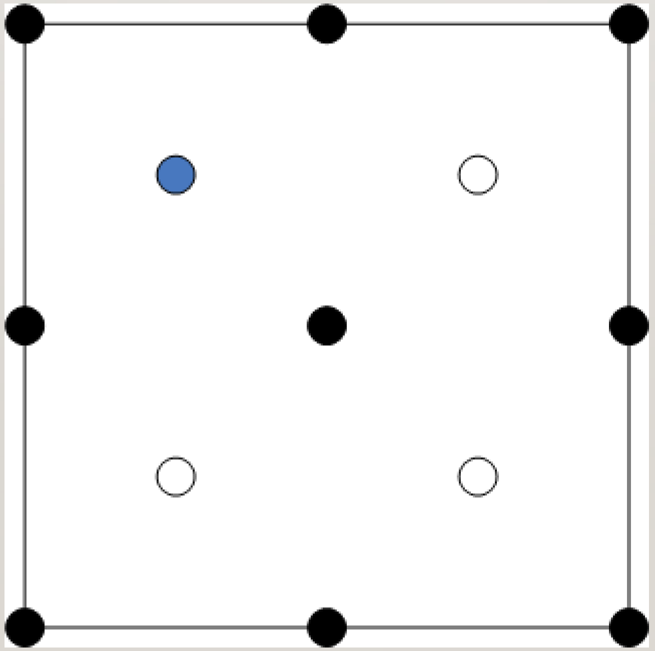
● Sample point

○ Validation point

🔵 Largest-error point

Adaptive Designs

Divide a cell with the largest error (here, the only cell) into 2^n sub-cells; put validation points in the middle of each sub-cells and evaluate the modeling error at all validation points:



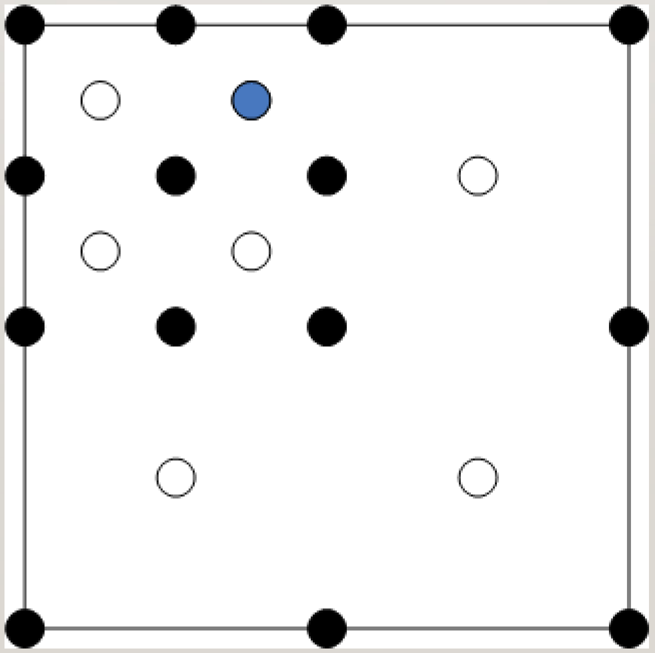
● Sample point

○ Validation point

● Largest-error point

Adaptive Designs

Repeat the procedure for the largest-error validation point:



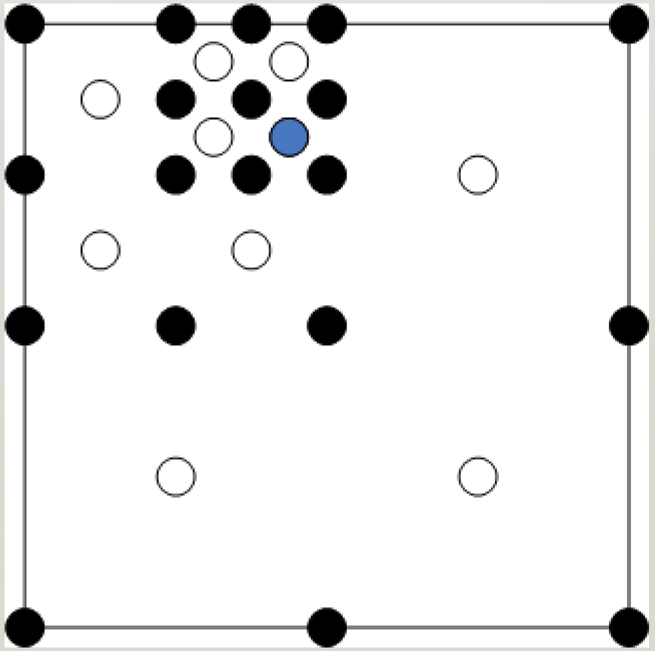
● Sample point

○ Validation point

● Largest-error point

Adaptive Designs

Repeat the procedure for the largest-error validation point:



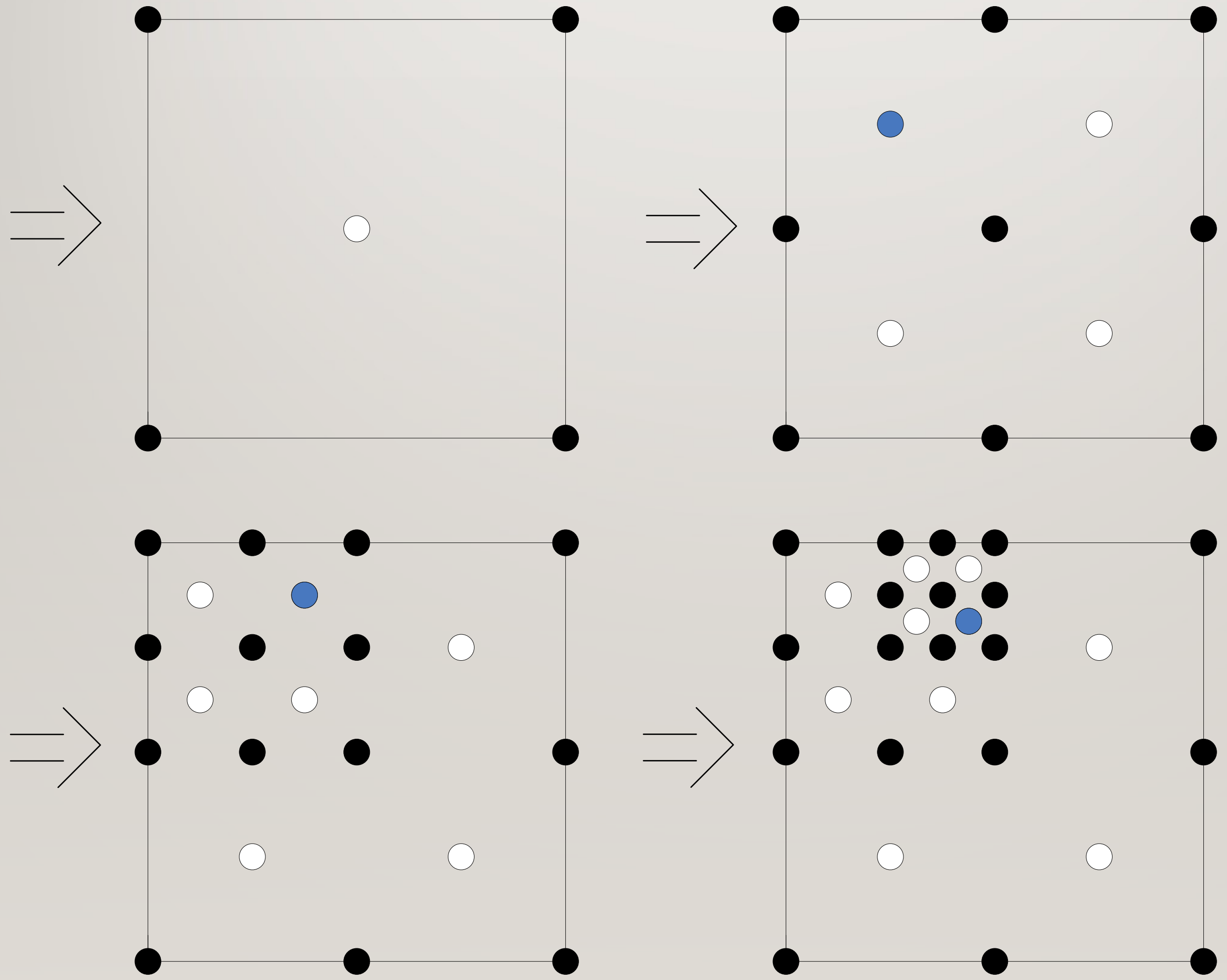
● Sample point

○ Validation point

● Largest-error point

Adaptive Designs

Summary of the example:



● Sample point

○ Validation point

● Largest-error point

Bibliography

A.A. Giunta, S.F. Wojtkiewicz, and M.S. Eldred, „Overview of modern design of experiments methods for computational simulations,” *American Institute of Aeronautics and Astronautics*, paper AIAA 2003-0649, 2003.

M.D. McKay, R.J. Beckman, and W.J. Conover, “A comparison of three methods for selecting values of input variables in the analysis of output from a computer code,” *Technometrics*, vol. 21, no. 2, pp. 239-245, May 1979.

J. Sacks, S.B. Schiller, and W.J. Weich, “Designs for computer experiments,” *Technometrics*, vol. 31, no. 1, pp. 41-47, Feb. 1989.

S. Leary, A. Bhaskar, and A. Keane, “Optimal orthogonal-array-based latin hypercubes,” *Journal of Applied Statistics*, vol. 30, no. 5, pp. 585-598, 2003.

K.Q. Ye, “Orthogonal column latin hypercubes and their application in computer experiments,” *Journal of the American Statistical Association*, vol. 93, no. 444, pp. 1430-1439, Dec. 1998.

K. Palmer and K.-L. Tsui, “A minimum bias latin hypercube design,” *IIE Transactions*, vol. 33, pp. 793-808, 2001.

J.R. Kalgnanam and U.M. Diwekar, “An efficient sampling technique for off-line quality control,” *Technometrics*, vol. 39, no. 3, pp. 308-319, Aug. 1997.

V.K. Devabhaktuni, M.C.E. Yagoub, and Q.J. Zhang, “A robust algorithm for automatic development of neural-network models for microwave applications,” *IEEE Trans. Microwave Theory Tech.*, vol. 49, no. 12, pp. 2282-2291, Dec. 2001.

Exercise 1: Design of Experiments - Auxiliary Functions

Implement procedure for evaluating the uniformity measure of the set of sample points in the design space as described during the lecture.

Implement visualization procedure that displays the sample points in two-dimensional design space $[l_1\ u_1] \times [l_2\ u_2]$.

Generalize visualization procedure so that it is able to display projections of the sample points in n-dimensional design space $[l_1\ u_1] \times [l_2\ u_2] \times \ldots \times [l_n\ u_n]$ onto two-dimensional sub-spaces $[l_k\ u_k] \times [l_l\ u_l]$, where k, l □ $\{1, 2, \ldots, n\}$, k □ l. Consider $n = 2, 3, 4$ and 5.

Test all the procedures using sets of sample points of your choice.

Exercise 2: Random Sampling

Implement random sampling algorithm that generates N points in n-dimensional design space $[l_1\ u_1] \times [l_2\ u_2] \times \ldots \times [l_n\ u_n]$.

Visualize the output of the algorithm for $n = 2$ and $N = 20$, 50 and 100.

Implement the algorithm for stratified random sampling (requirements as above).

Compare the uniformity of sample distribution for both algorithms for $n = 2$ and $N = 25$, 49 and 100 using both graphical visualization and the uniformity measure implemented in Exercise 1.

Exercise 3: Latin Hypercube Sampling

Implement the Latin hypercube sampling algorithm that generates N points in n-dimensional design space $[l_1\ u_1] \times [l_2\ u_2] \times \ldots \times [l_n\ u_n]$.

Visualize the output of the algorithm for $n = 2$ and 3 for the following values of N: 50, 100 and 200.

Exercise 4: Optimization-Based Sampling Algorithm

Implement the procedure for improving uniformity of sample distribution generated by LHS algorithm. Use *fmincon* routine from Matlab Optimization Toolbox to directly minimize the uniformity measure with location of the sample points as design variables.

Use your procedure to generate N points in n-dimensional design space $[l_1\ u_1] \times [l_2\ u_2] \times \ldots \times [l_n\ u_n]$. Compare the quality of initial and optimized distribution using both graphical visualization and the uniformity measure implemented in Exercise 1.